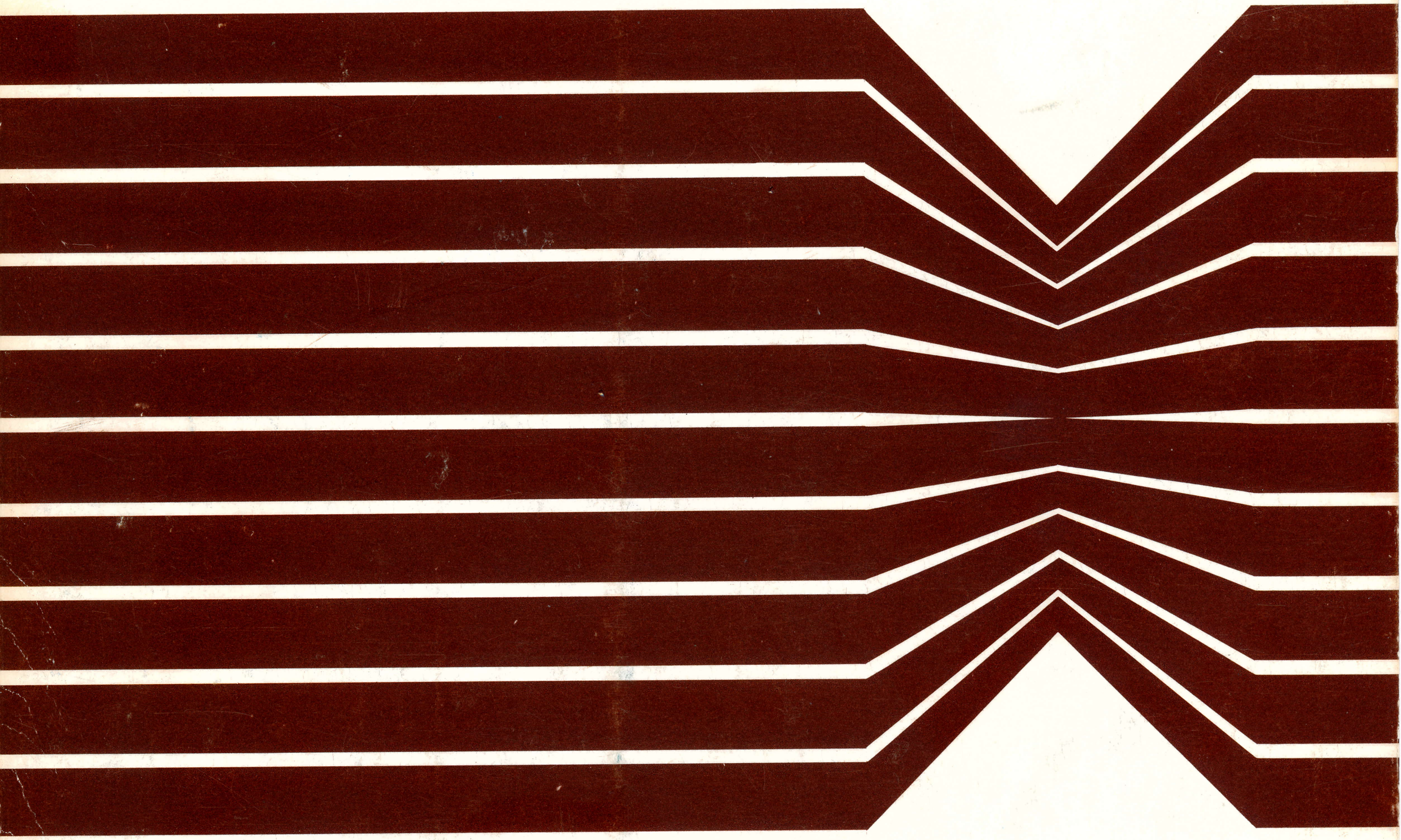
An APL Users Meeting

September
18, 19, and 20, 1978
Hotel Toronto
Toronto, Canada



Sponsored by:

I. P. Sharp Associates Limited

An APL Users Meeting

Toronto

September 18, 19, 20, 1978

Sponsored by:

I. P. Sharp Associates Limited

CONFERENCE PROCEEDINGS

DAY 1

MORNING
Session A: **Direction of APL Developments**

1. Sharp, I.P.
"The Evolution of Commercial APL Time-Sharing Systems" 1
2. Moore, R.D.
"A Data Communications System for the APL User" 5
3. Iverson, E.B.
"The Commercializing of APL\360 and some Plans for its Future" 13

DAY 2

MORNING
Session A: **Financial Planning**

1. Sakach, T.J.
"Corporate Financial Planning - is EDP Ready?" 22
2. Torrens, W.J.
"Evolution and Integration of a World Wide Data Collection and Reporting System"
3. Owen, D.N.
"Corporate Planning Systems for Non-EDP Users"

Session B: **APL in Exposition**

1. England, J.W.
"Calculus in APL. Why?" 27
2. Snyder, L.
"The Use of APL in Teaching Computer Science" 225
3. McIntyre, D.B.
"The Architectural Elegance of Crystals made Clear by APL" 233

Session C: **Applications in the Actuarial and Insurance Industry**

1. Oatman, J.E.
"A Discussion of the Practical Uses and Limitations of APL in Actuarial Research"
2. Trask, S.
"APL at Sun Life of Canada" 36
3. Hershfield, N.C.
"APL and the Wheel" 45
4. Thomson, D.J.
"Administration Applications for APL" 53

AFTERNOON
Session A: **In-house Migrations**

1. Lathwell, R.H.
   "SHARP APL as a Distributed Program Product" 264
2. Wilson, R.D.
   "Time-Sharing Information Management Service" 251
3. Elliott, M.
   "Migration to an In-house System" 255

Session B: **Graphics and Plotting Applications in Business and Industry**

1. Atutis, B.
   "Drafting with Computer Aided Graphics" 57
2. Hill, V.
   "APL Graphics for Trend Analysis and Planning" 63
3. Kemble, R.
   "Graphics for the Non-technical User" 70
4. Harrison, A.
   "An Interactive Hardcopy Graphics Package" 74

Session C: **Logistic Systems**

1. Fuller, J.J.
   "The Automated Inventory System" 92
2. Lapide, L.
   "Generalized Single Plant Freight Cost Model" 298
3. Payne, E.C.
   "Application of APL Modeling in a Petroleum Distribution Network" 97

DAY 3

MORNING
Session A: **Financial Applications in the Resource Industry**

1. Beagle, G.J.
   "Financial Modelling: Keys to Successful Implementation" 107
2. Graham, S.
   "Using APL to Help Make Decisions in a Hurry" 120
3. Martin, J.C.J.
   "Interactive Financial Applications in the Resource Area"
4. Wiig, K.M.
   "Cash Flow Planning System for an Integrated Oil Company in a Developing Country" 268

Session B: **Applications in Banking and Finance**

1. Boulay, R.A.
"Technical and Financial Evaluation of Capital Projects
- A Lender's Perspective" 134
2. Bettes, W.L.
"Financial Planning in International Banking" 153
3. Stein, G.R.
"The Process of Corporate Model Development in Banks"

Session C: **Quantitative Analysis and Statistical Techniques**

1. Loughran, B.P.
"Forecasting European Air Travel" 160
2. Heiberger, R.M.
"Variations on the Projection Function and their Application
to the Analysis of Variance" 173
3. MacLeod, N.C.
"Application of Polynomial Distributed Lag to Social and
Personal Costs of Unemployment" 348

AFTERNOON
Session A: **APL Standards for Documentation and Programming**

1. Contreras, L., K. Shahani and R. Skertchly
"A Conceptual Model for Interactive Systems"
2. Harrison, A.
"APL Standards and Documentation Within an Evolutionary
Environment" 193
3. Koutsouros, G.
"APL Programming in a Dynamic Planning Environment" 307
4. McIntyre, D.B.
"Experience with Direct Definition One-liners in Writing
APL Applications" 281
5. Iverson, K.E.
"Programming Style in APL" 200

Where page numbers are not indicated, papers were not available at time of printing.

LIST OF PAPERS BY AUTHOR

Atutis, B.P.	Drafting with Computer Aided Graphics	57
Beagle, G.J.	Financial Modelling: Keys to Successful Implementation	107
Bettes, W.L.	Financial Planning in International Banking	153
Boulay, R.A.	Technical and Financial Evaluation of Capital Projects - A Lender's Perspective	134
Elliott, M.	Migration to an In-House System	255
England, J.W.	Calculus in APL. Why?	27
Fuller, J.J.	The Automated Inventory System	92
Graham, S.	Using APL to Help Make Decisions in a Hurry	120
Harrison, A.	An Interactive Hardcopy Graphics Package	74
Harrison, A.	APL Standards and Documentation Within an Evolutionary Environment	193
Heiberger, R.M.	Variations on the Projection Function and their Application to the Analysis of Variance	173
Hershfield, N.C.	APL and the Wheel	45
Hill, V.	APL Graphics for Trend Analysis and Planning	63
Iverson, E.B.	The Commercializing of APL\360 and some Plans for its Future	13
Iverson, K.E.	Programming Style in APL	200
Kemble, R.	Graphics for the Non-technical User	70
Koutsouros, G.	APL Programming in a Dynamic Planning Environment	307
Lapide, L.	Generalized Single Plant Freight Cost Model	298
Lathwell, R.H.	SHARP APL as a Distributed Program Product	264
Loughran, B.P.	Forecasting European Air Travel	160
MacLeod, N.C.	Application of Polynomial Distributed Lag to "Social and Personal Costs of Unemployment"	348
McIntyre, D.B.	The Architectural Elegance of Crystals Made Clear by APL	233

McIntyre, D.B.	Experience with Direct Defintion One-liners in Writing APL Applications	281
Moore, R.D.	A Data Communications System for the APL User	5
Payne, E.C.	Application of APL Modeling in a Petroleum Distribution Network	97
Sakach, T.J.	Corporate Financial Planning - is EDP Ready?	22
Sharp, I.P.	The Evolution of Commercial APL Time-Sharing Systems	1
Snyder, L.	The Use of APL in Teaching Computer Science	225
Thomson, D.J.	Administration Applications for APL	53
Trask, S.	APL at Sun Life of Canada	36
Wiig, K.M.	Cash Flow Planning System for an Integrated Oil Company in a Developing Country	268
Wilson, R.D.	Time-Sharing Information Management Service	251

THE EVOLUTION OF COMMERCIAL APL TIME-SHARING SYSTEMS

Ian P. Sharp
I.P. Sharp Associates Limited
Toronto, Ontario

APL as a notation has been in the public domain for 16 years. The first serious implementation of it as a computer programming language operating in a time-sharing environment, was carried out at the T.J. Watson Research Center in Yorktown Heights, starting in 1965. The implementation was on an IBM 360 model 50, and the largest outside commercial user of that system was NASA at Greenbelt, Maryland. Various "bootleg" copies of the system were distributed to interested parties, mostly universities. The system took root specifically at the University of Alberta in Edmonton and Universite Laval in Quebec City, both in 1967.

The first commercial time-sharing version of APL outside of IBM was offered by the Marquardt Corporation of Van Nuys in California (now known as Proprietary Computer Systems) in late 1968. This was followed in 1969 by several other companies including APL Manhattan in New York, Scientific Time Sharing in Washington, The Computer Company in Richmond, Intermac in Rochester, Computer Innovations in Chicago, Cegos in Paris and I.P. Sharp Associates in Toronto. Of these companies, APL Manhattan and Intermac no longer exist; the remainder are still active and derive a significant proportion of their income from the sale of APL time-sharing.

One thing all of these companies had in common was that they all started with a Type 3 program product called APL\360 which was distributed free of charge by IBM. The system was remarkably stable and reliable, and it operated only under DOS. The consistency of the syntax facilitated the implementation of the interpreter, and the time-sharing system was simplified by the decision to use a fixed workspace size. The workspace size was 32K bytes, there was no support for any kind of file system, and the only terminals which could be used were the IBM 2741 and the IBM 1050.

As the result of an agreement between I.P. Sharp Associates and Scientific Time Sharing, various enhancements to the system were developed jointly by the two companies. Probably the major product of the co-operative agreement was the implementation of the file handling system by Larry Breed and Eric Iverson. Development of this was completed in mid 1970 and it is still substantially the file system offered by both companies today. From August 1969 to April 1972, both companies used the same computer facility located in downtown Toronto. During that period, the computer grew from a 256K model 50 to a 512K 370 model 145. In April 1972, STSC installed an almost identical configuration of 370 model 145 in Bethesda. As far as was practical, system developments by both companies were interchanged for a considerable period, but eventually differences in host operating systems reduced the value of such interchanges, and the practice ceased.

In 1973 IBM announced the availability of APLSV. By coincidence, that announcement was made here in Toronto at a meeting known as APL5. The new features of APLSV included the shared variable, ⎕*NAMES*, a new syntax analyzer and several other goodies, some of which already existed in commercial APL time-sharing systems, including our own. IBM had been under some pressure, particularly from the APL special interest group in SHARE to develop a file-handling system. As a result the shared variable processor was developed as a general solution to the problem of communication with other users and the host system. While the shared variable did not itself provide a file system, it did allow for orderly communication through the mechanism of an auxiliary processor, with an arbitrary existing file system.

Selling APL time-sharing to the world was in many ways simplified by the pioneering efforts of General Electric, who along with others, had established fairly well the commercial viability of time-sharing in general. The earlier development of Dartmouth BASIC had (and still has) a considerable following and thus the principles of time-sharing were defined and accepted, leaving only the problem of selling APL as a language. One of the interesting legacies of G.E.'s operation was the acceptance (almost universally) of a charging algorithm based on connect time, CPU time and storage.

In 1973, it was becoming obvious that the future of time-sharing (rather than the future of APL) lay in the use of the network by multi-office and multi-national companies. Up until that time, the tendency was to sell time-sharing to a single end user who could not afford a computer of his own, or alternatively, could not afford the high cost of duplicating an existing application system. Thus the early use of time-sharing was principally in the engineering area. Engineers were pleased with the fast turnaround they were able to get compared with that offered by traditional batch systems. Since engineers were largely users of Fortran, it is not surprising that BASIC and other derivatives of Fortran should have been so successful in a time-sharing environment. However, engineers are by nature extremely conservative, and the arrival of a new programming language such as APL was not greeted with much enthusiasm. In fact, engineers tend to be project oriented, and when caught in the middle of a project, are much too busy to learn something new, and when caught between projects, they simply have no requirement for a programming language of any kind.

Instead, the early users of APL time-sharing came from the business community. Largely they were planners and financial analysts with little or no experience with any programming language, people interested in building models of varying sizes and varying application areas. In retrospect and with the gift of hindsight, this is reasonable. APL appears well suited to any form of simulation and that, after all, is what planning and budgeting is all about. From the planning area the use of APL grew into other fields such as accounting, which we generally regard as traditional data processing areas.

It was these people, the planners and analysts, who then proceeded to recognize the value of a network. Their traditional problems were principally in the areas of data capture and consolidation together with the preservation of centralized control. A network solves these problems, and when combined with the facilities of a language that they liked, caused them to increase their use of the system at an alarming rate.

It is probably fair to say that the development of networks by all APL time-sharing vendors has been customer driven. At our customers' insistence, we embarked on a major network expansion in 1973 with the installation of a Transatlantic line, thus providing local dial access to users in Canada, the United States, and England. It was the expansion into England that brought into focus the shortcomings of time-dimension multiplexing which was the technology used by most people at that time.

In 1974 we were faced with the problem of replacing TDM technology either by relying on others to do the development or of doing it ourselves.

The decision was by no means clear cut. Companies specializing in communications were beginning to become established. In the United States, Datran, MCI, Telenet and Tymnet were developing precisely the technology which was of interest to us - it was obvious that the trend would spread to other countries. In Canada, the Trans-Canada Telephone System had announced a new digital network using packet-switching, following up its earlier (1973) offering known as Dataroute. Various government-owned European carriers were making similar plans and the question mainly revolved around the rather simple issue of whether all the carriers would be able to adopt a common protocol within a reasonable timeframe. At the same time, Michael Harbinson of our Dutch subsidiary, Intersystems B.V., was anxious to embark on the design and implementation of a low speed packet-switched communication system based on earlier work he had done in multiplexing Telex traffic. We decided that the prospects for international agreement on a packet-switched protocol were minimal and so we embarked on our own. A few months later in Geneva, an international agreement on the use of the protocol X-25 was announced. By that time the project was well under way and in late 1974, a primitive packet-switched system was operating between Amsterdam and London. Thanks to Roger Moore, this finally became a production system in late 1976 and since then, most of the cities in which we operate have been converted to it.

By late 1976, the majority of West European capitals were included in our network and the existence of the network itself became a catalyst in its usage. Within a year the total usage from Europe had grown to equal our Norh American usage.

As stated earlier, the growth rate of APL usage in a "converted" multi-national corporation is alarming, both to the buyer and to the seller. From the buyer's point of view it achieves a high degree of internal visibility, bringing with it some unfair comparisons with the performance of the internal data processing department. The cost of the network technology development could be economically justified by organizations such as our own, since these costs were effectively spread among several organizations, not one of which could justify the expenditures on its own. There also existed political and economic pressures to repatriate these applications to an in-house system.

From the vendor's point of view, the situation was also undesirable, since large quantities of computer and human resources were being used at cut rates, as the buyer's demands for volume discounts grew. The dependence of a small company such as our own, on a few large customers, is clearly undesirable since it leads to instability.

We were asked on numerous occasions (in fact these questions still persist) as to why we did not install a processing facility in Europe since the volume of our business there would appear to justify it. Aside from the obvious question of communication costs compared with the costs of a manned computer facility, there appeared to be no logical method of dividing the files on a geographic basis. There is at least one user in each city in which we operate who has a requirement to communicate with, or share programs and data with, another user in at least one other city in which we operate. Thus, the common access to a system is at a customer level, not at a geographical location level. It was also apparent that although most of our customers were experienced in the running of large scale computer systems, most had little experience in the running of large scale communication systems.

It was this observation which led us to the idea of installing our version of APL on a CPU owned and operated by a customer on his premises, while at the same time maintaining access to the communications network for that customer's mainframe. Our initial thinking was that the customer's mainframe should operate in a manner identical with our own; that is to say, under the same operating system (a modified form of DOS) and in any case as a dedicated system supporting no other language processors. This approach works well only with those customers whose usage is large enough to warrant a dedicated computer together with dedicated operations, systems programming and management staff.

From there it was a short step (philosophically but not practically) to the idea of distribution of the program product to run under the customer's favourite operating system which was presumed to be one of the standard IBM supported systems, such as MVT, VM, SVS or MVS. This policy was announced in late 1977 and appears thus far to have had a favourable reception.

We have, from time to time, addressed the question of our long term future as a vendor of APL time-sharing. If we make the assumption that APL usage will continue to grow, how do we provide the computer resources to cope with it; is it through a CPU in each city or wall to wall CPU's in one huge data centre? Can we rely on improvements in computer technology to provide us with ever increasing power in a single CPU and ever increasing capacity in disc storage? What we have decided to do is to limit the size of the time-sharing service to that which can be supported by two, and two only, CPU's. The degree of expansion that this permits is still substantial since we expect and indeed can rely on the fact that advancing technology will improve CPU speeds. The reason for two CPU's is probably obvious; it provides for backup, albeit degraded and we already have the technology for operating two CPU's as a single unified system. The expansion beyond this will come in a rather unusual form of distributed processing, in which the software supporting the time-sharing system will be installed on customer operated mainframes, which in many cases may be larger than the mainframes we ourselves operate.

The generally accepted view of distributed processing involves the reprogramming of existing systems in order that they function on a smaller dedicated machine. The costs of software development and maintenance has led us to the conclusion that the generally accepted ideas on distributed processing are fundamentally wrong. Similarly, network architecture which assumes that data should be transferred to the CPU on which the user program exists, appears to be in error, since it makes much more sense to transfer the program to the CPU which has access to the files. Such transfers of programs can be carried out effectively only if the two machines in question are compatible.

A very fruitful area of development has been the implementation of large scale data bases with specialized access methods. In the main, these data bases comprise information in the public domain, some of which has hitherto been available in computable form. The first such data base was one consisting of stock market trading data from the Toronto Stock Exchange which was made available in 1970. Since then it has expanded considerably to include other North American exchanges, options, commodities, etc. In 1972, David Keith started work on the aviation data base consisting of U.S. airline data filed with the Civil Aeronautics Board. To make reasonable use of this data, he also implemented a language, now known as Magic, to access, manipulate and display this data. Magic has undergone several changes and continues to develop. It became our standard method of accessing the economic time series data published by Statistics Canada, the National Bureau for Economic Research, the International Monetary Fund and the Organization for Economic Co-operation and Development. We intend to continue this development in terms of the facilities available within Magic and also the number and types of data bases that we will support. As an example, we have recently mounted the American Petroleum Institute data on imports and movements of petroleum products and the Bureau of Mines data on natural resources development.

In other areas we are currently supporting major developments in pension evaluation, lease analysis and financial statement consolidation.

As with communications, the direction of our development is largely in the hands of our customers. One of the main purposes of this conference is to hear from you, the user of APL, about the features you would like to see in the language, additions or changes you would like to see in the system, facilities you would like to see in the communications network, data bases you would like to see us support and program packages you would like to see developed. We hope you will not be reticent in expressing your views.

A DATA COMMUNICATIONS SYSTEM FOR THE APL USER

Roger D. Moore
I.P. Sharp Associates Limited
Toronto, Ontario

Introduction

SHARP APL began operation in the summer of 1969 with a network which contained one time division multiplexor and sixty dial ports. By early 1976 this had grown into a medium-sized collection of time division multiplexors including one trans-atlantic link. During 1976 and 1977 this network was replaced by a packet-switched network. The primary goal of the network, both before and after the change, has been the provision of APL service to remote lowspeed terminals.

When introducing new technology it is desirable to remedy the drawbacks of the old without sacrificing its benefits. This general goal resulted in some specific, attainable objectives. It also revealed areas where compromise was required. This paper attempts to record various observations and decisions made while building a data communication system for the the APL user.

User Interface

The network should not be particularly obvious to the user. Initial connection is kept as simple as possible. With a single input, the user can identify both terminal type and destination APL machine. If the requested APL facility is available, user interaction with the network ends. After the connection has been established, the network attempts to act as a simple forwarding agent: output from APL is passed to the terminal for printing, and input from the keyboard is passed to APL. Unlike certain public networks there is no provision for escape to the network during a session. The closest approach to this is the network recognition of XON/XOFF when the terminal is not in the "input to APL" mode. These control characters allow the terminal to control the rate at which output is sent from APL to the terminal.

The network requires a certain amount of information about the user's terminal. This information is required for terminal control and for printing legible error messages. The APL\360 trick of examining the initial right parenthesis to resolve BCD vs Correspondence has been retained. This character was chosen on the grounds that many people would accidentally type it during a sign-on attempt and the information required by the network would be acquired. The initial right parenthesis from an ASCII terminal is not actually used by the network. It is retained briefly so that an obfuscative blot can be elicited from APL after a connection has been established. It is also used as the left character of the destination selection command. The ASCII character set issue is largely avoided by the absence of special characters in messages from the network.

TDM Drawbacks

Prior to the installation of the present packet-switched network, I.P. Sharp Associates (IPSA) used older technology to accomplish a similar function. Some of the deficiencies of the time division multiplexor (TDM) technology hastened the conversion to a packet-switched technology.

Telephone lines are a known source of transmission errors. This is particularly true when efficient modems are used. An efficient modem provides a transmission rate (measured in bits per second) which exceeds twice the bandwidth (measured in hertz) used. Although this gives a rather efficient use of the analog voice channel, a small number of errors (perhaps one hundred per hour) are introduced. With the TDM technology of several years ago, most of these random errors tended to be visible to the APL user. Erroneous data would be printed on the terminal or sent to APL. Some method of protecting the user from minor errors on the network trunks was required.

The usual method of compensating for short error bursts is to group several characters into a block and to append some redundant bits to the block. The receiver can then perform some arithmetic operation upon the received block including the redundant bits to see if a certain class of error has occurred. If the arithmetic operation in the receiver indicates an error, then the receiver requests retransmission of the block. The Sharp network uses the same redundancy check as does HDLC (x*16+x*12+x*5+1 residue). This gives the ability to detect all error bursts of less than seventeen bits and most bursts of greater length.

The network does not provide any protection on the lowspeed link between the terminal and the network. In some locations this continues to be a source of error. Control of this error source requires some sort of retry protocol between the network and the terminal. Little work has been done in this area by IPSA.

Simple TDM technology dedicates one channel to every port passing over a particular trunk. The channel assignment is made at installation time. It cannot be altered without briefly removing the multiplexor from service (or in one extreme case returning it to the factory). This fixed assignment of bandwidth can impose limitations on the total number of ports in areas where digital bandwidth is not free. When the entire terminal population consisted of 134.5 bps terminals of the same character length, this was not a particularly onerous restriction. It was possible to squeeze about fifteen 134.5 bps channels into a 2400 bps highspeed trunk. The same 2400 bps can only accommodate eight 300 bps channels.

During the period of transition from 134.5 bps nine-unit terminals to the present 300 bps ten-unit terminals, autospeed presented a further complication. It is desirable to have a single telephone number for both kinds of terminals. With simple TDM technology it was necessary to have separate telephone numbers and separate channel groups for the two terminal classes. By using a technology without fixed bandwidth allocations the transition from 134.5 bps BCD to 300 bps ASCII was eased.

Packet-switching has allowed a bit of gambling on bandwidth requirements. The total dial-up capacity at a remote site can be arbitrarily large. There is no "a priori" bandwidth allocation for a particular port. Bandwidth is allocated on demand exactly as CPU time and file bandwidth are allocated within APL. If an overload exists along a particular route, the symptom is a general slowdown of output. This is quite noticeable to the APL user as printing will occur in bursts corresponding to individual packets. This may be caused by a general overload situation such as when trans-atlantic traffic is running over a single trunk rather than the normal two. An alternative cause is a transient overload due to a trunk going out of service for a second or two. If this happens during a peak period a transmit queue can build up which will take several seconds to clear down to an innocuous level. Even under light load conditions, a brief service interruption on a trunk will exhaust the endnode buffer pool for a printing terminal. The resulting delay will be noticed by the user.

The location of the pause (mid-line rather than inter-line) is a side effect of the method of packing APL output lines into network buffers. The educated user can use the location of the printing pause to discriminate between poor performance in the APL machine and network degradation. The observed network degradation may be caused by either an acute or chronic overload. Output is held at the APL machine (or in network transmit queues). This was considered preferable to loss or corruption of output, and it has also influenced pricing. A 120 cps terminal has a lower priority on transmission from APL than does a conventional terminal. This may reduce the mean print speed observed at the 120 cps terminal to less than 120 cps. For this reason, imposition of a connect time surcharge on 120 cps terminals seemed rather unfair.

If a fixed bandwidth had to be assigned to every dial-up port, it would be difficult to accommodate the number of ports demanded by the typical optimistic branch manager. With dynamic contention for bandwidth, a relatively large number of ports can be installed at a site to accommodate peak connect loads. There is a gamble that not all ports will be continuously engaged in printing. A terminal which is not being used for output requires zero bandwidth.

Packet-switching Drawbacks

After concluding that simple TDM technology was unacceptable because of the problems mentioned above, some choices still had to be made in a packet-switched environment. Although TDM technology does have certain drawbacks it also has certain user benefits. If all of the allocated ports are being used to print full speed it is quite efficient. It can almost always provide very good response time.

Most data transmission systems require some extra bits which are not directly useful to the end user. For example, the asynchronous terminals used with most time-sharing systems have start, parity and stop bits associated with each character. None of these is directly useful to the end user, but they are required by the asynchronous protocol. Similarly, a TDM requires some extra bits to maintain framing synchronization. A packet-switched system is even worse. It has a redundancy check for each block. Each block also contains addressing and sequencing information to allow it to be delivered to the proper port in sequence. With the present Sharp network packet size, about fifty percent of the bits transmitted are overhead bits of no direct value to the user. With a TDM, the overhead content is closer to twenty percent. The dynamic bandwidth allocation capability of the packet system tends to balance the higher efficiency of the TDM system. Thus, the number of 30 cps terminals which can be comfortably accommodated in the packet system is only slightly higher than the number for a TDM system (twelve versus eight assuming 2400 bps trunk).

Packet-system efficiency is theoretically easy to alter. The number of overhead bits per packet is fixed for a particular protocol. By increasing the number of useful bits in a packet while preserving a constant overhead, the apparent bandwidth efficiency (useful bits ÷ total bits) can be increased. Unfortunately, the need for acceptable response time constrains packet size.

Delivery of a packet from APL to an endnode may require forwarding the packet down more than one link. The time delay imposed by one link (`T[I]`)is given by the formula:

```
T[I]← CD[I] + LENGTH÷BPS[I]
```

`CD[I]` is a constant delay associated with the link and its modem. A typical value for an intracontinental link is between 10 and 20 msec. `BPS[I]` is the link speed in bits per second. `LENGTH` is the packet length in bits. If the subscript vector `PATH` selects the links required to reach a specific endnode, the delay to the endnode is `+/T[PATH]` or

```
(+/CD[PATH]) + LENGTH × +/÷BPS[PATH]
```

The network currently uses a 256-bit maximum length packet. For a 2400 bps link, the length sensitive term in the delay is 106 msec. Thus the observed delay with one or more 2400 bps links in the path is rather sensitive to packet length. Doubling the packet length would change bandwidth efficiency from the present .5 to around .75 but it would also increase delay by one or two hundred milliseconds in certain parts of the network.

A possible solution to the capacity versus response tradeoff is to vary the packet size with the terminal speed and location. A more complex solution would be to use short packets for the first two packets of an APL response and then revert to a larger packet length.

One important design consideration in a packet-switched system is the most likely form of use. For an APL-oriented network, the expected form is point-to-point communication with relatively long holding times. A single port with a connected physical terminal is associated with a particular APL terminal task (T-task). Any interaction with other ports in the network will be via APL facilities for intertask communication, rather than through network facilities. Not all computer networks have this point-to-point property. In particular, a process control system which monitors many different sensors and sends commands to many devices does not have this point-to-point property.

A network with an orientation toward point-to-point communication is often referred to as a virtual call network. It resembles a switched telephone system in which physical circuits are established and held to accommodate a particular point-to-point call. Certain administrative packets are exchanged to set up a virtual call; this precedes the transmission of any data. In the the Sharp network this occurs after the initial input line (normally "o)") and before the security blot is printed. After the call has been set up, an abbreviated form of addressing can be used. This is possible because there is little need to identify the originator of a packet within a virtual call. There are only two ends to a virtual call; the end which receives a packet assumes it came from the other end.

Addressing and Routing Considerations

An alternative form of organization is the datagram network. In a pure datagram network every packet includes the name of the packet originator as well as the destination. In theory this allows any port to conduct simultaneous dialogues with all other ports in the network. For some applications this can be useful, but time-sharing does not seem to be one of those. Datagrams are used within the Sharp network for certain administrative purposes such as call setup and status monitoring. It appears that these uses could be replaced with virtual calls with only moderate inconvenience. Converting the Sharp network to a strict virtual call organization would provide certain simplifications. In particular, the amount of control information required to route datagrams is proportional to the square of the number of nodes in the network. In a pure virtual call network, routing information need only be computed and maintained for existing rather than potential virtual calls. This reduces the quantity of routing information to the number of virtual calls times the mean path length. As the size of the network is currently around sixty nodes and continually increasing, datagram addressing is becoming rather unwieldy. Changing to a pure virtual call network appears necessary for further growth.

Another advantage of a fixed-route virtual call system is that one of the more unpleasant theoretical properties of a store and forward network can be eliminated. Most retransmission and queuing schemes do not provide an upper bound on packet delivery time. It is also possible in some systems to lose a packet without notification being provided to the sender. The usual approach to the lost administrative datagram problem is for the sender to wait a finite time for a response. If no response is received within the timelimit, the sender assumes (perhaps incorrectly) that the original packet or the response has been lost.

With virtual call addressing, a unique call number is assigned to every virtual call in the system. By constraining all traffic to a point-to-point format over fixed routes, two problems can be solved. The packets for a particular virtual call will be stored in the forwarding queues of a small identifiable set of nodes. This is not true for elaborate dynamic routing systems. When the call is completed or aborted, the packets of the call can be removed from the network. For a call which is being completed in an orderly fashion, the precaution of procrastinating the call completion signal until the data packets have been forwarded is adequate. When a node or a link along the route of a virtual call fails, packets are assumed lost and the virtual call must be aborted. An abort signal is passed from both sides of the failure to the endpoints of the virtual call. As the abort signal passes along the route it has the side effect of destroying any packets belonging to the virtual call. The virtual call number is then available for reuse, as the previous generation of packets with the same virtual call number has been destroyed.

One other drawback to the existing scheme of routing by node number is that traffic passing between a pair of nodes reachable by independent paths is constrained to a single path. One reason for this constraint is the simple route table organization. Within each node the route table is stored as a vector indexed by destination node. Each element of the vector gives the direction in which some other node lies. As the current route table entries are scalars, it is impossible to encode an alternate route. A more serious problem is the mechanism which aborts virtual calls after a link failure. It is extremely important that both endpoints of an aborted virtual call receive notification of the abnormal call termination. When this mutual termination requirement is combined with a requirement that calls not using the failed link remain undisturbed, the termination and routing algorithms impose a condition upon the route tables. The route from A to B can be expressed as an ordered list of intermediate nodes. With the present routing algorithms, a necessary and sufficient condition for mutual call termination after link failure is that the route from node A to node B must be exactly the reverse of the B to A route for all node pairs. When every virtual call has its own route, the consistency requirement only applies to individual virtual calls. The route from end A to end B must be the reverse of the B to A route for that virtual call. A simple forwarding validation algorithm can guarantee this in realtime. Other virtual calls with identical endpoints can be routed independently.

Link Management

One area of concern in the initial design was maintaining performance in the presence of minor line errors. A trans-atlantic link running at 7200 bps can store four or five unacknowledged packets. Some link management protocols such as X.25 require retransmission of the entire pipeline after a single error. This has the advantage of preserving packet order over a single link. It has the drawback of exaggerating the effect of a single error. Instead of retransmitting the single corrupted packet, four or five packets may be retransmitted. The Sharp network retransmits only the corrupted packet. This has the drawback of permuting the packet order along the link. If the various packets in the pipeline were associated with independent virtual calls, this would be of no consequence. The order within a virtual call could be disturbed by retransmission. In a network which uses multiple routing, differing transmission delays along distinct paths can give the same effect. The possible ill effects of order loss are avoided by including an interend sequence number in every packet belonging to a virtual call. The receiving endpoint can then process the packets in the order in which they were emitted rather than the order in which they are received.

APL Interface

The network would be of little value without a connection to the APL machine. Several methods of connection were discussed during network design. An obvious solution would be to preserve the symmetric hardware of a TDM system. At the computer room end, data from a single virtual call would materialize on a physical port attached to the network. This physical port would be electrically connected to a Memorex 1270 terminal controller. This solution is sometimes called lowspeed demultiplexing. It is simple in concept and appears exactly like an ordinary 1270 configuration to APL. In addition to the obvious problem of requiring large quantities of hardware, lowspeed demultiplexing has other drawbacks. The worst is the way in which the entire system is slaved to a 300hz oscillator in the terminal controller. Data transfer to and from APL is constrained to a single procrustean speed. There are situations where transfer at a rate which is either faster or slower than the normal terminal speed is desirable.

Running the traffic into the APL machine on a single highspeed line and demultiplexing within the APL machine was contemplated. This was rejected mainly because of the excessive changes which would have been required in the APL software.

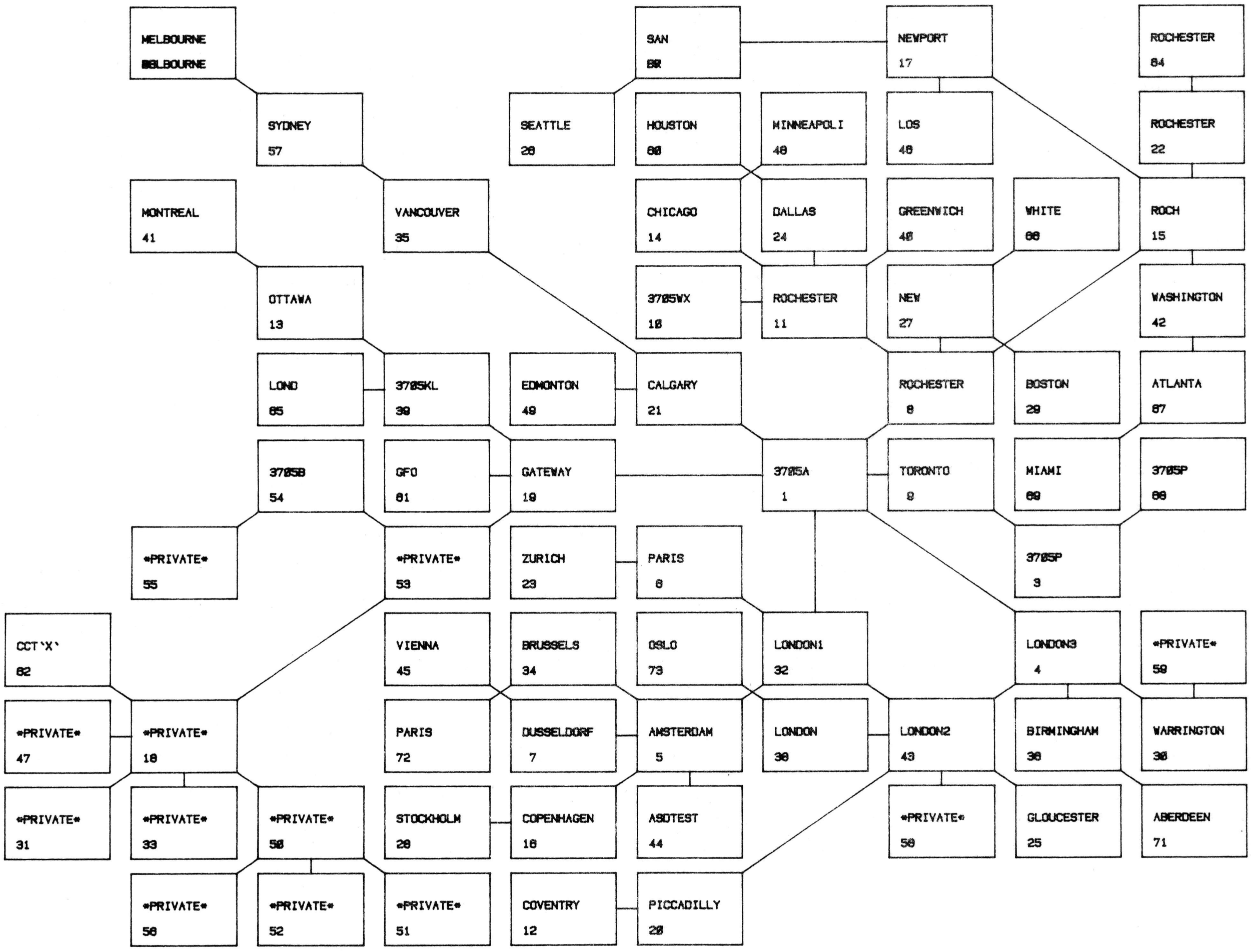
The solution which is in use now involves placing a stored program computer between the APL machine and the network. This is a true frontend computer as it is attached directly to a 360 byte multiplexor channel. On the channel side, it looks very much like a MRX 1270 or IBM 2703. Every virtual call uses a separate address on the byte multiplexor channel. The frontend computer can trivially adjust the rate at which data is transferred from APL to the network. This adjustable rate is fairly important in achieving acceptable response time. When APL starts generating output in response to an input, all of the network buffers for that virtual call are empty. In this situation, data is transferred at high speed over the byte multiplexor channel to fill an initial buffer ration. Data transfer for that virtual call then stops until a buffer has been sent to the terminal. Another buffer is then filled with data from APL and sent through the network. This process occurs outside of APL and is not usually visible. The network buffers are independent of APL buffer size and APL output line boundaries. If the network can absorb output faster than APL is generating it, a partial buffer will be sent. A typical case is the *SENT* response to)*MSG*. When APL is generating output more rapidly than the network is willing to accept it (⎕←⍳1*E*4), the normal APL mechanism for suspending an output-limited task is used.

Flow control is also useful on input to APL. SHARP APL has retained the 1969 APL\360 scheme of demand assignment of input buffers. The default assumption in APL is that an input line will be quite short. If an input line is longer than expected, extra buffers will be assigned within APL. It is theoretically possible for the network to present data faster than APL can allocate buffers. To avoid loss of keyboard input, a method of regulating the rate at which the frontend computer presents data to APL was required. This was accomplished by sending an explicit end-of-buffer signal from APL to the frontend computer whenever an APL buffer is filled. The frontend computer suspends data transfer for the affected virtual call until the APL machine signals that a new input buffer is ready. This signalling is normally effected by command chaining on the byte multiplexor channel and thus takes less than sixty microseconds. If intervention by APL software is required to allocate a new buffer, the "buffer ready" signal may be delayed for an arbitrary period. The data is held in the frontend computer until the "buffer ready" signal is received from APL. This solution avoided the potential embarrassment of carefully forwarding (and perhaps retransmitting) a packet over the ocean, and then corrupting it with a race condition in the computer room.

Establishment of a new call is another area where a frontend computer was useful. The number of ports in the network somewhat exceeds the number of T-tasks which can be simultaneously supported with the present IPSA hardware configuration. This makes dynamic allocation of byte multiplexor channel addresses desirable. When the frontend computer signals a new connection to APL, control information pertaining to the call is transmitted to APL. The terminal protocol (ASCII vs BCD), terminal speed, port location and the user's right parenthesis are passed to APL. The APL software needs all of the information except port location to communicate with the user's terminal. The port location is not directly used by the APL system, but is available for various applications (particularly operational statistics) which need to know the geographical location of a terminal.

The diagram below illustrates the topology of the network. Each rectangle represents a node in the network and each line represents a link. About sixty of the nodes shown are currently operational. The diagram includes some planned installations and some nodes used for hardware and software testing. The only redundant link in the network is in the 1 4 43 32 loop in the lower right-hand region.

The program which draws the diagram normally uses an APL terminal as an output device. This places some constraints on the diagram (particularly on line direction). The version shown here was drawn by an HP7221 plotter with assistance from Bob Bernecky.


MELBOURNE MELBOURNE
SAN 88
NEWPORT 17
ROCHESTER 84
SYDNEY 57
SEATTLE 28
HOUSTON 80
MINNEAPOLI 48
LOS 48
ROCHESTER 22
MONTREAL 41
VANCOUVER 35
CHICAGO 14
DALLAS 24
GREENWICH 40
WHITE 68
ROCH 15
OTTAWA 13
3705WX 10
ROCHESTER 11
NEW 27
WASHINGTON 42
LOND 65
3705KL 38
EDMONTON 49
CALGARY 21
ROCHESTER 8
BOSTON 29
ATLANTA 67
3705B 54
GFO 61
GATEWAY 19
3705A 1
TORONTO 9
MIAMI 69
3705P 60
PRIVATE 55
PRIVATE 53
ZURICH 23
PARIS 6
3705P 3
CCT'X' 62
VIENNA 45
BRUSSELS 34
OSLO 73
LONDON1 32
LONDON3 4
PRIVATE 59
PRIVATE 47
PRIVATE 18
PARIS 72
DUSSELDORF 7
AMSTERDAM 5
LONDON 38
LONDON2 43
BIRMINGHAM 36
WARRINGTON 30
PRIVATE 31
PRIVATE 33
PRIVATE 50
STOCKHOLM 28
COPENHAGEN 16
ASDTEST 44
PRIVATE 58
GLOUCESTER 25
ABERDEEN 71
PRIVATE 56
PRIVATE 52
PRIVATE 51
COVENTRY 12
PICCADILLY 20

Conclusion

Conversion from TDM technology to packet-switched technology has been done with minimal change to APL. Both the internal structure and the external appearance are approximately as they were before the introduction of packet-switching. There has been some sacrifice of response time in order to achieve greater reliability and capacity. The greatest present and future benefit is the ability to provide reliable service in many different geographical locations.

THE COMMERCIALIZING OF APL\360 AND SOME PLANS FOR ITS FUTURE

Eric B. Iverson
I.P. Sharp Associates Limited
Toronto, Ontario

In the Beginning

In the summer of 1969 we received a tape from IBM containing a very impressive piece of software. It was called APL\360 and it was free! APL\360 was to be run on our IBM 360/50 to provide APL time-sharing service to be marketed by both I.P. Sharp Associates (IPSA) and a newly formed U.S. company called Scientific Time Sharing Corporation (STSC). Under the guidance of Roger Moore (IPSA) and Larry Breed (STSC, now with IBM), the system was soon installed and operational. The sureness of the installation and the immediate high reliability that was achieved (not too surprising as Roger and Larry were two-thirds of the team that wrote the software) set the standards in commercial time-sharing that the company has worked hard to maintain.

APL Only

One of the earliest key decisions we made was to run a dedicated APL time-sharing service. We decided not to offer access to other languages for two primary reasons: (1) we couldn't excel at everything, so we decided to try to excel at APL, and (2) APL was the language with a future. A major benefit from this single language approach was that we could run DOS, the simplest, most efficient, and most reliable of the IBM operating systems. In most installations the operating system chews up not only most of the machine resources, but also most of the system programming resources. Our time could be spent providing improvements visible to the end user rather than dealing with the software labyrinth created by thousands of IBM programmers. For us, the maintenance of DOS is a critical but part-time job. The one significant addition to DOS was the addition of POWER as a spooling system in early 1974. This was done essentially on the insistance of Bob Bernecky over the protest of nearly everyone else. Today, with the growth we've seen in background jobs interfacing with APL, it is clear we could not survive without POWER. At first, implementation and functional deficiencies severely limited the types of applications for which our service was appropriate. In the early days we often had to say, "we can't do that now, you're better off doing it at another service bureau in batch COBOL". Rather than drastically dilute our efforts by struggling to provide COBOL to the user, we instead examined **why** our APL service was inadequate to a particular application and then would proceed to provide the facilities so that the next time we could say, "sure we can do that in SHARP APL, forget about COBOL".

The File System

The need for a file system within APL, and the ability to move data between APL and the outside world was clearly perceived and documented while APL was still an experimental system at IBM. It was obvious that for APL to be successful in solving problems in the commercial environment that a file system was required. Before APL\360 was installed, we had agreed that a file system was of fundamental importance, and immediately after APL was offered as a service, Larry Breed and I started the design and implementation of the APL file system. Already we were behind the demands and requirements of our first customers. While Larry and I worked night and day on the file system, other programmers were forced to provide stop-gap measures. The first interface with the outside world was a crude mechanism which allowed the system card reader to be signed on as a terminal and hence able to read cards into a workspace. The subsequent "refinement" on this was a utility which could read cards and put them into a workspace which was then saved. With this utility we loaded our first "large" volumes of data for customers. In those days large meant more data than you'd care to type at a terminal. This provided an interface with the outside world, but the only way to store data was still the saved workspace. The first public data base was implemented using these limited capabilities. Each day we received a deck of cards containing closing prices on the Toronto Stock Exchange. The cards were read into a workspace which was then saved. The appropriate workspace from the "data base" was then loaded, the data copied in and processed, and the workspace resaved. The data base consisted of about 100 saved workspaces, each containing a few days' data. There was no *⎕LOAD* or *⎕LX*, so when you wanted data you had to manually load the correct workspace and start typing.

Finally, in the summer of 1970, after nearly a year of commercial service, the file system became available. Users now had convenient access, under program control, to arbitrarily large amounts of data.

As background programs (written in Assembler, COBOL, etc.) could access the file system in the same way an APL user could, the file system also gave APL its first clean, efficient, and well defined interface with the outside world.

Shortly after the file system became available we attended an historic APL conference sponsored by NASA, at which Larry presented a paper on the file system. The interesting point is that several other papers, from other commercial APL time-sharing companies, were presented describing their new file systems. Although the differences were what we, as designers, found interesting, the historical importance is that we all perceived this to be the most important area to work on to make APL viable in the commercial world. It is safe to say that the significance of APL in commercial time-sharing today is due largely to the early availability of file systems.

Later in 1970, in cooperation with the Financial Post, the Toronto Stock Exchange data base (closing prices in workspaces) was replaced by a much larger base of data maintained in our first really large, public data base. The Financial Post data base has continued to grow and now covers most North American exchanges as well as a great deal of other financial information.

Formatting

One of the first pieces of public library software was the APL function *DFT* which was used to format data for printing reports. The earliest users of APL had been satisfied with simple unformatted output. For the commercial user, elaborate formatting was not only essential, but it represented a major cost in his use of APL. Considerable pressure was applied for cheaper, and equally important, more powerful formatting. To solve this problem, *⎕FMT* (at that time, before distinguished names, it was spelled *ΔFMT*) was implemented. Like many enhancements, *⎕FMT* has undergone several improvements and extensions since its original implementation, and now with the use of *⎕OUT* and HSPRINT, formatting is reduced to its most convenient form and is free. It's hard to imagine a commercial APL service without a powerful and efficient formatting primitive.

The Remote User

With files, *⎕FMT*, and a continually improving system, some remote users still had (and still have) a significant problem. The user, with a convenient and reliable link through a terminal, has the system at his fingertips. Unfortunately, a terminal is not suitable for moving large amounts of data to or from the system, and when faced with that problem, the remote user is painfully aware of how slow planes, courier services, and the mails are. Peter Teeson's work in HSPRINT provides cheap and efficient printed output on the highspeed printers at the computer centre. For customers in Toronto, and for reports whose usefulness are not too diminished by a day or more delay in delivery, this is fine. We have attempted to cope with the more general problem by using standalone computers with tape drives and communication capabilities. With these standalone computers, a separate network that is somewhat in conflict with our terminal network, and considerable ingenuity in remote IPSA and user offices, we have managed to provide some timely movement of large amounts of data. In short, the current facilities are inadequate. What is needed is a network, integrated with the terminal network, that as directly as possible links the user's applications in the system to remote devices (printers, tape drives, CPUs, etc.). Roger Moore's group in communications is doing considerable work in this area now, and they are talking of significant new capabilities next year.

The Customer Is Always Right

Our commitment to improving the system so that it can effectively handle an ever broader range of data processing problems makes the development group very dependent on customer input. We are not dependent on users for detailed specifications of new primitives, but we do look to them for applications which are too expensive, awkward, or even impossible to do in SHARP APL. The job of the development group is to determine the underlying facilities needed to make impractical applications practical, and then to provide those facilities. It is interesting to note that IPSA is its own largest user, and that members of the development group spend more time programming in APL than they do in Assembler.

Customer input to the development group falls into four categories: cost, time, productivity and capability. Users want existing applications to cost less and they want to be able to run new applications for which they calculate the costs to be prohibitive. They want to be able to get results from existing applications more quickly, and they want to be able to do new applications which will take longer to run than they have time to wait. They want increased programmmer productivity (particularly from less experienced people). And finally, they want to be able to do things which just aren't possible with the current system. We have done development in all four areas, but the developments that are most significant are those which are effective in more than one area. The file system, and *⎕FMT* clearly addressed all four areas. The next three sections discuss developments that were responsive to customer needs and address all four areas of customer requirements.

N-tasks and B-tasks

The introduction of N-tasks and B-tasks in early 1976 addressed the problem of the rapid increase in the number and importance of "production systems". These are applications which are run periodically (daily, weekly, on demand, etc.) and which produce their results without the tinkering or ministration of a programmer or terminal user.

A terminal is a joy while developing an application, but is unnecessary, and is in fact detrimental to a "production system". By late 1975 it was not uncommon to find APL terminals with signs on them to the effect - RUNNING PRODUCTION SYSTEMS. PLEASE DO NOT DISTURB. I AM IN THE COFFEE ROOM. The production clerk, with fingers crossed against a line hit, was waiting for the terminal to be free so the next production job could be started. The other aspect of this form of production (a terminal had to be used, and had to be watched) was that production systems were being run during the heavy prime time load on the system. The system was often swamped at peak times and quite idle during off hours. To solve these problems we introduced N-tasks and B-tasks. Their acceptance was slow at first. It seemed that users who loved the highly interactive nature of APL were slow to realize they could now have their cake and eat it too (i.e. they could develop and debug interactively, but the application, once in production, could be run in a traditional and advantageous batch environment at considerably reduced cost). Now, with experience, N-tasks and B-tasks are an indispensible tool in running large production systems. In retrospect, it is hard to see how we managed without them. We have reached a level of maturity in their use where the next round of ideas for improving them are starting to come in. In particular, several people have requested that some N-tasks be scheduled similarly to B-tasks (particularly regarding restart), that *⎕OUT* be extended to capture all terminal input and output, and that *⎕IN* be provided to allow "terminal" input to an application to be provided from a file.

PACKAGES

Unlike the slow acceptance of N-tasks and B-tasks, PACKAGES were accepted overnight. Not only were they used immediately in new systems, but old systems were rewritten so as to benefit from the considerable simplification and reduction in costs. The importance of PACKAGES lies in two distinct areas. The first is that they significantly increase the richness of APL data structures (and the power of APL lies in its richness of primitives, operators, and data structures). The second is the important realization that there will always be applications larger than a workspace can handle. There are applications running on our current workspace size that consist of more than a million bytes of functions. PACKAGES, along with the file system, provide the necessary structure for dealing with large applications. A great deal of thought and effort went into the design of PACKAGES, with many months of discussions, arguments, proposals and counter proposals preceding the final draft of SATN-14. Once the final draft was agreed to, though, Doug Forkes had it implemented and available within a month. More recently, David Allen has added some new package primitives and increased the efficiency of some cases for the old ones.

The success of PACKAGES is in large part behind our recent decision to start serious design work on enclose (denoted by <) and disclose (denoted by >) primitives, thus introducing generalized arrays as a new data structure.

Recovery and Restartability

An area that has increased in importance as the number of production systems has increased is crash recovery and restartability. The problem is twofold: to reduce the amount of work lost in the event of a system crash, and to make it possible for applications interrupted by a crash to ensure their integrity and to restart processing. The workspace recovery facility which became available in early 1974, and eliminating delayed writing of data by the file system in April of 1976 (SATN-16) did a great deal to reduce the loss of work or data in the event of a system crash. For relatively simple applications they also provided sufficient capability to allow successful and reliable restarts. The general problem of restart, though, remains untouched. Although we don't have clear ideas in this area, we do recognize its importance and will provide enhancements in this area.

Growth: Quantitative and Qualitative

Growth is a significant customer input to the development group. Certainly as much work has gone into making growth transparent as has gone into providing new or more efficient capabilities. The system is continually being asked to service more users, with more applications, accessing more data. Several times now, in several different areas of the system, the growth has not been simply quantitative, but instead has crossed some boundary and become qualitative. Quantitative changes can usually be addressed by more or faster hardware, whereas qualitative changes require significant changes in software. In short, the original APL\360 and the file system were never designed to cope with 200 or more simultaneous users, accessing billions of bytes of shared data and functions. This is particularly true of the schedulers, I/O managers, and the organization of system directories. All of these areas, for both APL and files, have undergone one or more major rewrites to be able to cope with the growth. When we fall behind, either qualitatively or quantitatively, the customers see bad response time and poor throughput.

In 1973 the growth was such that the only viable solution was to split the system in two. This we did, and ran a geographically split service with one 370/145 servicing the United States and another 370/145 servicing Canada and Europe. At first this was an operational nightmare. We had all the problems of two systems, complicated by the fact that there was considerable requirement for moving workspaces and files back and forth between them. When the system was split, the utilities for dumping and retrieving workspaces and files could only be run while APL was shut down. This meant workspaces and files could only move from one system to the other in the narrow window at midnight. We were soon forced to develop utilities that could run while APL was running. The development of the on-line File Utility was a major project and was done very well, under considerable pressure, by Peter Wooster. It has been subsequently enhanced by Gary Dennis to the point where it has completely replaced the old, dedicated utility.

It is interesting to note that merging the systems together again did not diminish the importance of these on-line utilities. We now run far more dumps and retrieves than could possibly be done in the narrow window when APL is shut down. They also allow operations to run customer's jobs in a much more timely manner and are a prerequisite to running APL uninterrupted round the clock. With the availability of local access in Australia, shutting the system down from midnight to 2 am Toronto time (mid afternoon in Australia) became a major embarrassment. At the same time there was increasing pressure from several customers for 24-hour service. The only requirement for shutting down at midnight was to run the daily File System backup program which wrote changed file components to tape. Now that Leslie Goldsmith has finished the design and implementation of a File Backup utility that runs while APL is running, we are finally able to schedule APL available round the clock.

In early 1975 it was finally decided to replace each of the overloaded 370/145s with a 360/75. The 360/75 was fast enough that we could merge the systems together again, giving us a back-up system and a development machine. At this point we did the analysis of running a merged system and not too surprisingly discovered that several parts of the system, already strained by a single 370/145 load, could not possibly cope with the combined load. The major projects were a complete redesign of the APL scheduler by Leigh Clayton, and the introduction of another system directory level for the file system by Peter Wooster. Finally, in September, 1975, the systems were merged. The merged system provided better response time and allowed unification and simplification of our two communication networks. Most importantly it allowed our growing business with multi-national customers to proceed without concern as to whether the various branches and the data they needed were on the same machine or not.

Having run split systems once, we were determined not to do so again. Late in 1975, work was started on developing software to allow the two machines to be coupled together so that both could service a single user base. This effort was sufficiently successful that we were able to support a unified system load on the two machines considerably in excess of what one machine could have supported. Eventually, of course, the two 360/75s were overloaded and one of of them was replaced by the considerably faster AMDAHL V6.

At the end of this paper are two graphs that are interesting indicators of growth. Note that the chart "Processing Power Running SHARP APL" does not include the fact that we've conservatively achieved an overall increase in productivity of a factor of three in the base software. We expect this trend to continue; however, the growth will be primarily outside of our own computer centre, and will be at customer installations running in-house SHARP APL. The chart "WORKSPACE SIZE" also indicates what will be a continuing trend. Demands for larger workspace size are always very high before we are able to justify the necessary software and hardware changes. In particular, software was more of a problem than hardware. Thanks to Leigh Clayton, the software problem is now solved and I would expect a somewhat more rapid growth in workspace size over the next several years than we have seen over the past nine. But there will always be applications larger than the workspace size can handle. The experience gained in the last few years with paging systems (i.e. systems which erase objects no longer needed and bring in packages of new objects from file) is pointing the way to a system-provided "automatic paging system". A sketch of such a system would be that before reporting WS FULL, the system would automatically erase objects from the workspace that could be recovered when needed from file, and when an object was needed, and it wasn't in the workspace, it would be automatically brought in from file.

Development Resources

The merged system on a single 360/75 gave the development group, for the first time, a development machine. Up to this time development testing was restricted to a few hours each night after midnight and to slightly more generous times on weekends. The availability of this resource, in conjunction with the maturity of the development group (a core of seven people with more than 30 years of APL implementation experience) led to a very rapid rate of development which has been maintained. This rate of development forced the introduction of the SATN documents. To quote from SATN-0:

> "SHARP APL is a dynamic environment that has been steadily improving and expanding for several years. It seems the ability to extend the APL environment has always outstripped the ability to document the extensions. Sometimes new facilities are released promptly, but with little or no documentation. Other times their release is considerably delayed waiting for proper documentation. The intent of SHARP APL technical notes (SATNS) is to fill the gap between "no documentation" and "good documentation" and to minimize the time between "implemented" and "released". Following the publication of a SATN or of a series of related SATNS will come a proper manual complete with extensive examples and guidelines."

The SATN index, with release dates and revision numbers, gives a good indication of the rate and direction of developments visible to the customer. A parallel set of documents, SINS, cover changes to the system not directly visible to users.

Now with the AMDAHL V6 as the main production system, the remaining 360/75 is generally available as a development machine. However, when not in use by the development group, it is coupled to the AMDAHL.

People

The history of serious commitment to APL, the success in developing and delivering a commercial APL service, and in large part the high visibility of the development group starting in late 1975 led to Paul Berry (in late 1976), and Dick Lathwell and Joey Tuttle (in late 1977) joining the company. These three gentlemen had all worked for APL inside IBM before joining IPSA. Dick Lathwell was the other third (along with Roger Moore and Larry Breed) of the team that wrote APL\360. He was also the main designer of Shared Variables and is currently finishing an implementation of Shared Variables (with some interesting extensions) for SHARP APL. Paul Berry, who wrote some of the earliest APL educational material, is becoming the source of the manuals "complete with extensive examples and guidelines" promised by SATN-0. Joey Tuttle was responsible for some of the very large APL applications within IBM, and is now applying that experience in the commercial world.

Doing it in APL

As stated earlier, much of the programming done by the development group is done in APL. Some of this is in customer application areas (e.g. Bob Bernecky's, Peter Wooster's, and Dave King's work in graphics; Doug Forke's work in text editing and interfacing with an automatic typesetting system; and Leslie Goldsmith's work in MAILBOX and WSDOC). But most of it is more closely related to running and developing a time-sharing service. The complex billing, invoicing and usage systems were all written entirely in APL by Dave Markwick. More recently, a sophisticated analysis system of access violations was done in APL by Mike Symes, the newest member of the development group. Every access violation (e.g. file access error, incorrect workspace lock, incorrect sign-on lock) is recorded by the system. These records are analysed as they happen and the violation system reacts immediately to suspicious patterns, and provides daily summary reports for the operations manager, Bill Apsit. APL systems are used to maintain all source code, as aids in debugging and dump analysis, and for modeling (e.g. the File System was completely modeled in APL before the first Assembler instruction was written). It has progressed to the point where development that would have previously been done at the Assembler level, is now done in APL. A good example of this is the B-task scheduler, which is a complex task scheduler that traditionally would have been done at the Assembler language level. Instead, by using a privileged interface to the system, Leslie Goldsmith wrote it in APL. This trend of system software being written in APL is one that will continue.

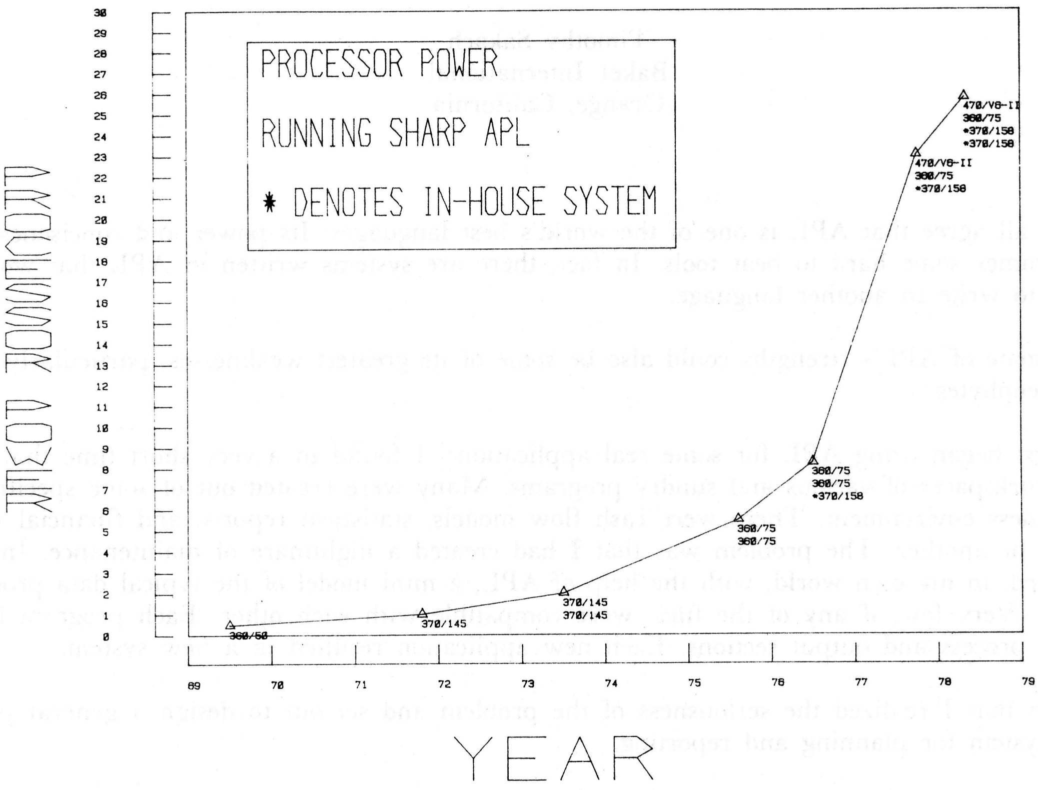
The Future

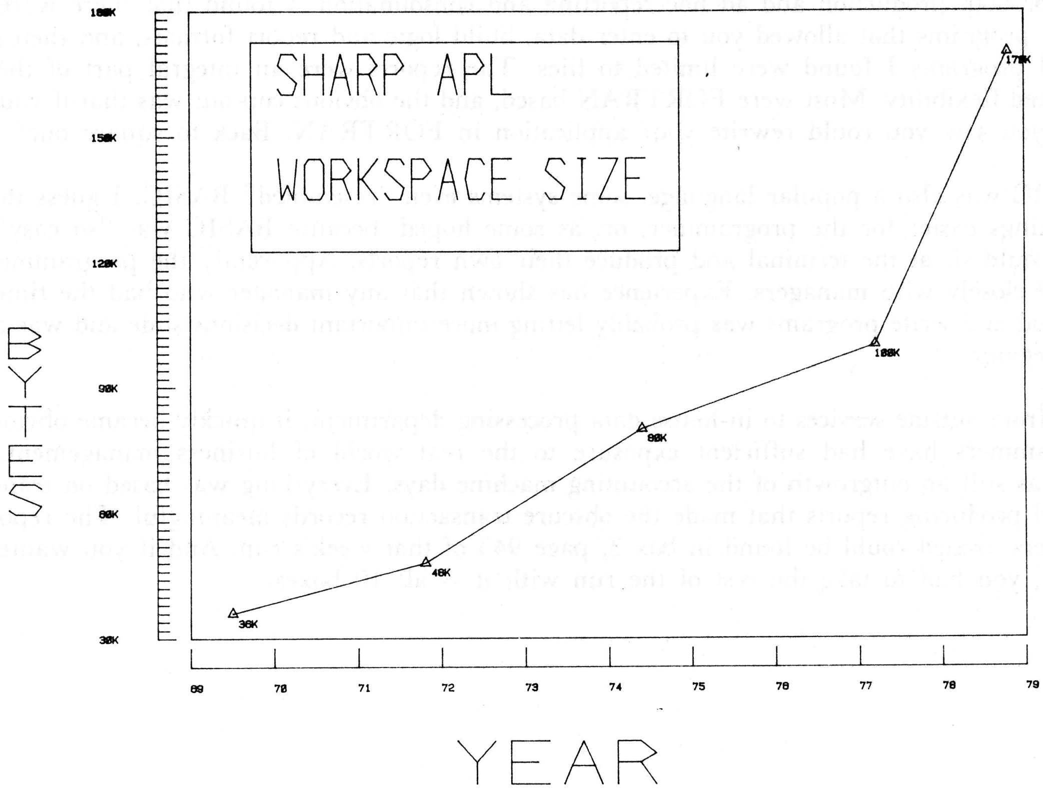
The development group is stronger, in both numbers and experience, than ever before. We have increasingly better facilities and tools at our disposal, and foresee our software running in a large number of in-house installations. To say the least, we are excited about the future.

The list of projects to do, and ideas to study, grows faster than projects are completed. A few of these projects were mentioned earlier and I'd like to add error trapping, interrupt control, new primitives, and new operators (in the long run, this is a most significant area) to that partial list.

The APL language and the SHARP APL system are evolving rapidly and the next few years will see more change than the last ten. If we are careless, the evolution will lead to complexity without any appreciable increase in power. But, if all of us in the APL community are quite careful, the evolution will lead to simplicity, elegance, and power.

APL is a language with a future (as we decided a decade ago) and we are looking forward to participating in that future.


PROCESSOR POWER
RUNNING SHARP APL
* DENOTES IN-HOUSE SYSTEM
PROCESSOR POWER
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
360/50
370/145
370/145
370/145
360/75
360/75
360/75
360/75
*370/158
470/V6-II
360/75
*370/158
470/V6-II
360/75
*370/158
*370/158
69
70
71
72
73
74
75
76
77
78
79
YEAR


SHARP APL
WORKSPACE SIZE
BYTES
180K
150K
120K
90K
60K
30K
36K
48K
80K
108K
170K
69
70
71
72
73
74
75
76
77
78
79
YEAR

CORPORATE FINANCIAL PLANNING - IS EDP READY?

Timothy Sakach
Baker International
Orange, California

I think we all agree that APL is one of the world's best languages. Its power and conciseness offer the programmer some hard to beat tools. In fact, there are systems written in APL that would be impossible to write in another language.

However, some of APL's strengths could also be some of its greatest weaknesses, particularly in the hands of neophytes.

When I first began using APL for some real applications, I found in a very short time that I had over 100 workspaces of various and sundry programs. Many were created out of some specific need in the business environment. There were cash flow models, statistical reports, and financial reports of one sort or another. The problem was that I had created a nightmare of maintenance. In short, I had created, in my own world, with the help of APL, a mini model of the typical data processing department. Very few, if any of the files, were compatible with each other. Each program had its own input, process and output sections. Each new application resulted in a new system.

It was then that I realized the seriousness of the problem and set out to design a general purpose operating system for planning and reporting.

Before I began the actual work on what was later to become the Baker Planning System (BPS), I set about on a systematic search for any available software that might in a general way encompass the need for both production and ad hoc reporting and consolidation. I found that there were many application programs that allowed you to enter data, build logic and report formats, and then get the results. All programs I found were limited to files. The reports were an integral part of the logic, which limited flexibility. Most were FORTRAN based, and the obvious cop-out was that if you didn't like what you saw you could rewrite your application in FORTRAN. Back to square one!

Since BASIC was also a popular language, some systems even "resembled" BASIC. I guess that was to make things easier for the programmer, or, as some hoped, because BASIC was "so easy", even managers could sit at the terminal and produce their own reports. Apparently the programmers had not worked closely with managers. Experience has shown that any manager who had the time to sit at a terminal and write programs was probably letting more important decisions slide and was at least a poor delegator.

Switching from outside services to in-house data processing department, it quickly became obvious that few programmers have had sufficient exposure to the real world of business management. Most software was still an outgrowth of the accounting machine days. Everything was based on transaction records and producing reports that made the obscure transaction records meaningful. The report that the managers needed could be found in box 3, page 943 of that week's run. And if you wanted only that report, you had to take the rest of the run with it -- all 10 boxes.

Another problem existed that gave management some additional cause for alarm. It sounded like a good idea when the data processing department separated accounting systems from manufacturing systems. At last they had someone from the EDP department who was specifically assigned to a specific management area. What wasn't told was that the programmer/analyst was working independent of inventory, accounting, payroll and order entry. And worse yet, none of the other programmer/analysts were aware of what anyone else was doing. They were given a box to play in. You can imagine the chagrin that came over some managers when they asked for summaries of all related aspects of the business from inventory to payroll. The system was not designed to do this and the data structure for each application was different from any other. A terrible conversion problem had to be solved. However, management was assured that DP used structured programming and used the "top down approach", and were using on-line methods for programming and testing. In addition, over the past year DP had added the newest compilers and some fancy programming aids guaranteed to make programming easier and less costly. No one seemed to care about data structures. It never dawned on some data processors that 90% of the data in a corporation is in some way related.

This unfortunate situation is not meant to represent any particular company. It represents a "state of affairs" that currently exists in many companies. Surveys have shown that 80% of all companies are dissatisfied with their data processing departments. That leaves only 20% who are doing a really good job serving their companies.

So where does BPS come in?

BPS was born out of a need; a need that was not being met by time-sharing companies, not by in-house data processing departments, and not by personal computers.

BPS was designed to be a **reporting, consolidation** and a **planning** system. To fulfill these objectives, it requires a **network** and **data base management system**, and a system for entering and editing data, logic structures, and report formats. It was designed to be used by the user departments and requires at least one individual in that department to be trained to run the system.

The system consists of a network, provided by I.P. Sharp Associates, a data base management system which uses the file subsystem and the package capability of SHARP APL, and a general purpose operating system.

The GPOS does not take on the peculiarities of a particular industry segment. It can be used by individuals to do personal financial planning. It can be used by marketing people to build forecasting and econometric models. It can be used for budgeting, for income statement and balance sheet analysis, for cash flow analysis, and for manpower planning. It is not restricted to time series analysis, as most systems are. Its most important feature is that it distinguishes between the problems that the system must solve and the problems that belong to the user.

BPS was designed to assist all levels of management. Operational management is particularly interested in details and in projects. They may be more concerned about meeting time schedules and quantities. Their budget planning generally reflects a dollars per hour relationship.

At the other end of the extreme is corporate management whose primary concern is the financial well-being of the corporation. They may look at the relationship between number of employees and the amount of revenue generated, and the amount of capital working to produce that return. They are also interested in changes, both favorable and unfavorable.

To give you an idea of the generalness of BPS, it is currently serving both extremes of management. This is due to the similarity of data at both the operational level and at the corporate management level. The common factor is that both can be expressed numerically -- one in hours and the other in financial numbers like **sales, inventories**, etc.

We have designed a data base system for managing numerically-related data. The data is stored in an easy-to-understand form of a matrix with numerical row and column identifiers.

The system limits itself to dealing with numerically-related data. No attempt has been made at this time to process transactions or to create data files of alphanumeric data.

Guidelines for System Design

There are some aspects of the system design which I feel would be helpful to anyone who is considering designing major systems in APL.

Control the Workspace

In major systems, your workspace is one of your most valuable resources. Certain practices, such as argument passing, raveling, reassignment to the same variable, and parentheses, if not handled with discretion, can be workspace killers. In every feasible case we tried to avoid unnecessary duplication of data which occurred automatically with the operations mentioned above.

Program for the Machine

Without the assistance of function editors and documentation aids, it is very difficult to build optimum APL statements. We used the diamond separator as much as possible, particularly in operations that might involve looping back to the beginning of the line. The diamond also saves on storage and on CPU time. Because APL uses two-step processing translation and interpretation, we took pains to avoid unnecessary retranslation of APL code. In addition, we made every attempt to avoid penalizing the user who does things right by not including error messages on the same line. When error trapping becomes available we will be able to optimize even more.

Use N-tasks

N-tasks were used in every possible situation -- sending reports to a printfile, running major consolidations, and running logic pages to perform calculations against many data files. Macros available to the user to be included in the logic pages allow N-tasks to be initiated automatically, whenever the user wishes.

Simplify User Programming Efforts

Every APL program starts from zero-base. You start with a clear workspace and build from there. You must build your own input/output process. BPS takes care of input and output, file control, and almost every other tedious problem facing the user. In addition, it gives the user some easy-to-use tools for entering and editing data, creating logic, controlling consolidation, and formatting output. It permits the user to build complicated ad hoc reports in the kind of timeframe management expects. It is possible to achieve 1 day turnaround in applications with all data on-line and readily available. PL1 and FORTRAN programs that may have taken nearly 6 months to program and debug and test were replaced with applications on BPS that required only 1 or 2 days.

Control Costs

One of our primary objectives was to produce a system which took care of the difficulties in programming and at the same time, cost no more to the user than if the user had to build it from scratch. That is, we gave the power to the user that he expects from a computer system, but with no penalty for using the system. In fact, tests have shown BPS to be more economical for the user for two reasons -- optimized system coding and faster turnaround time for the user.

Use Data Base Management Techniques

Most data base specialists agree that it is not really feasible for a corporation of any size to try to control all of its valuable data in a single data base. They feel that there could be hundreds of data bases, but all of them created and controlled by a single system -- a data base management system.

For example, there are many companies in Baker with dissimilar problems. Each has to treat its operational data in a special way. This is most noticeable between the service and manufacturing companies. However, because Baker is a planning company, the data reaching divisional, group, and corporate levels must be compatible. We must be able to consolidate data, and BPS has assisted Baker in achieving unity and compatibility out of diversity. This is made possible by a data management system in BPS. Even though each division may have specially tailored financial and operational models, a simple user-built program in BPS extracts the needed corporate data and sends it to corporate management. The objective of the system was not only report generating, but also report **reporting** .

Potential Problems

Peak Loading

It is possible for over 100 users in a single corporation to be using BPS simultaneously. Deadlines and schedules must be met. This forces most of the corporation to be on at one time. We hope the system will be ready for this kind of loading. Users are encouraged to get on-line because BPS is so easy to use. They begin thinking of applications they never considered before. During the first month of operation, the corporate office alone had four projects demanding more or less equal time and every terminal was in use.

Demands on the File Subsystem

BPS has done away with proliferation of workspaces. One data base could contain hundreds of applications operating on hundreds of data files within that and other data bases; and when you combined that with the peak loading problem mentioned above, it is easy to see some unusual demands being placed on the file subsystem.

Highspeed Printing

Through BPS, it is easy to generate reports consisting of hundreds, if not thousands, of pages. Highspeed printing capability is much in demand, as is the demand for 1200 baud phone lines. Work is underway to improve the printing of reports, and at the time of this writing, research is being done to develop a good CRT and printer combination.

Summary

Management seems to be the last to be served by effective data processing. The reasons are somewhat clearer now as to why this is so. Data processors thought they were serving the management of the companies by assisting in the control and management of operational data. At the same time they lacked the facility to meet the direct needs of financial and corporate management. Innovations in communications, languages and terminals were needed to meet this heavy demand. Even though part of the job was done by smaller systems and by special-purpose application programs, much more work needed to be done.

Any system directly serving management has to be:

- Flexible
- Quick
- Effective
- Efficient
- Easy to use
- Powerful
- Available
- General

BPS was created, not in a data processing shop, but in the corporate office. Even though it meets most of the qualities mentioned above, it will improve through time and usage. SHARP APL has made a significant contribution towards meeting the needs of management. Without the features of SHARP APL, BPS would not even be half the system it is today. EDP is ready to take on the understated, underestimated, ignored and rejected areas of international corporate planning and reporting.

CALCULUS IN APL. WHY?

James W. England
Department of Mathematics
Swarthmore College
Swarthmore, Pennsylvania

Abstract

This paper considers an example of a mathematical topic presented in APL. This topic was presented at different times to two mathematics classes. The level of APL knowledge of the two classes was quite different. The results of this experience indicate that in order for APL to be used as a notation for teaching mathematics, it is necessary for the level of APL knowledge on the part of the students to be well above the required mathematical sophistication.

Calculus in APL. Why?

Since writing the title of what I thought would be in this paper, I have found it necessary to change the content of the paper. Two events occurred since that time which have caused that change. The first event is the lack of a completed study of the calculus course in APL which was to become the subject of this paper. Candor forces me to admit that it was not a breakdown of some computer. Ah well, we all long for the good old days of unreliable computers.

The second event is a sequence of discussions I have recently had with a number of former students of mine. These students were divided between those having taken calculus and linear algebra in APL and an advanced course in linear algebra and several variables calculus in APL. The topic of these conversations was, "is APL an effective notation for teaching mathematics?"

The idea of using APL as a notation for mathematics has been discussed since the first days of APL. There is, however, not much experience available on this proposition. The lack of much experience in this area is to some extent a result of a relatively few number of mathematics texts written in APL. At this time, I know of only one calculus text in APL and three or so precalculus texts. A more important reason for the relatively small amount of use of APL as the notation for teaching mathematics is, I think, that changing the notation of a mathematics course is such a fundamental change that most people would prefer to take some middle road and use APL as a computer language in an ancillary way instead of making the more drastic change. Thus, my experience in the two courses may be of some value. The specific experience I want to report on is what I think is a successful use of APL as a mathematical notation in Calculus and Linear Algebra course and a less than successful use of APL in this way in the more advanced course. While there were probably many reasons for these events, my conversations with former students in these courses have led me to focus on one as being central. An apparent requirement for the effective use of APL as a notation in teaching mathematics is to keep the level of sophistication of APL roughly equivalent to or above the level of difficulty of mathematics. That is, to use APL in this way the students must, if having no prior knowledge of APL, be at the start of a subject whose ideas initially use little in the way of APL. Otherwise, it seems necessary to spend a significant amount of time teaching APL and then begin teaching the subject. (By a significant amount of time I mean more than the first week or so of a fourteen week semester.)

The problem of keeping the level of APL commensurate with the level of mathematics was mentioned in passing in a previous paper of mine [England 1]. However, I now think it to be much more important than I did at that time.

In this paper, I want to present a topic, in outline form, which is covered in both the introductory and advanced course which illustrates the problem under discussion in a rather clear manner. The topic is that of finding the general solution to a homogeneous systems of equations. In the introductory course, this topic is covered about the second week of the second semester, i.e., after about sixteen weeks of using APL, while in the advanced course it is covered in the third week of the first semester. In both courses we devote the first week to formal instruction in APL and then discuss APL in only a secondary way throughout the remainder of the course.

The reactions to this topic in the two courses could not have been more different. In the introductory course, most of the students seemed to be able to follow the discussion with the result that they could implement the technique and understand why it worked. In the advanced course, most students could not understand why this technique worked and could only implement it in routine examples. My explanation for this is that in the introductory course, the students had, by the time we discussed the topic, come to a deeper understanding of APL than the level of mathematics presented while in the other course students were struggling with the APL to such an extent they could not see the mathematics because of the notation. (I should note that the advanced students are so designated because they have completed single variable calculus in High School.)

The explanation I will give follows the one given in most linear algebra courses. It comes after students know a bit about subspaces of Euclidean space and basis. It is, of course, essential that students learn well how to find a general solution to a system of equations since this technique forms the basis of many of the succeeding topics in the course. As indicated above, this description is in outline form. The discussion I give in presenting this to a class would have many more examples and explanations of specific points. I presume that the reader of this explanation has either a good understanding of APL or of the mathematics described.

We are given a matrix *M*, called the matrix of coefficients and we wish to find all vectors *V* so that `(ρV) = ¯1↑ρM` and `∧/0=M+.×V`. Of course, the set of all solutions of the system of homogeneous equations `∧/0=M+.×V` forms a subspace of K dimensional Euclidean space where K = `ρV`. Thus, by a general solution to this system of equations we mean a matrix *B* with the property that the columns of *B* are a basis of the solution space. That is, the columns of *B* must be linearly independent and have the property that if *W* is any solution of the system of equations then there exists a vector *C* with the property that `∧/W=B+.×C`. That is, *W* is a linear combination of the columns of *B*.

Recall that there are two distinct steps involved in solving this problem. The first is to reduce the matrix of coefficients to a simple form (reduced row echelon form) and the second is to find a general solution to the reduced system, and hence the original system.

Each of these steps involves knowing when a matrix is in reduced row echelon form. A matrix *M* is in reduced row echelon form provided; (1) the first non-zero element in each row is a 1 and comes to the right of the first non-zero element of each row above it; (2) if a column contains a first non-zero element of some row then all of its other elements must equal zero; (3) all rows containing only zeros must be below any row which contains some non-zero element. It sounds like this definition was written by the Internal Revenue Service - which is a testimony to the clarity of common language. We now give this definition in APL. We do so by defining a function *RREF* whose argument is a matrix and whose result is a 1 if the argument is in reduced row echelon form and a 0 otherwise. We need to identify the indices of the first non-zero element in each row. The following expression does this for us. `D← +/∧\M=0`. (Throughout, we assume `⎕IO←0`.) Notice that if some row of *M* has only zeros, then the corresponding entry in *D* is equal to `¯1↑⍴M`. If we let `C← (⍳¯1↑⍴M)∊D` then if *M* is in reduced row echelon form `C/M` should have exactly one 1 in each column and all the other elements should equal zero. That is, with these definitions *M* is in reduced row echelon form if and only if the following expression is a 1; `∧/∧/(C/M) = (⍳1↑⍴M)∘. = ⍳+/C`. Thus, an adequate definition of *RREF* is:

```
      ∇ Y←RREF M;D;C
[1]     D←+/∧\M=0
[2]     C←(⍳¯1↑⍴M)∊D
[3]     Y←∧/∧/(C/M)=(⍳1↑⍴M)∘.=⍳+/C
      ∇
```

A couple of examples should make the definition clear.

```
        M                                  P                                  N
  1 ¯3  0  0  4                        1 3 0 0                            1 3 4 0
  0  0  1  0 ¯2                        0 0 0 0                            0 0 1 0
  0  0  0  1  0                        0 0 0 1                            0 0 0 1
  0  0  0  0  0                        0 0 1 0                                  RREF N
        RREF M                                 RREF P                     0
1                                    0
```

The reason for wanting the matrix of coefficients in reduced row echelon form is that the associated systems of homogeneous equations is as simple to solve as possible.

We now define a monadic function, *EASYSOLVE*, with the property that if its argument is a matrix, *M*, in reduced row echelon form then its result is a general solution to the system of homogeneous equations `∧/0 = M+.×V`.

The first step is to remove all rows of *M* which contain only zeros. These correspond to equations which are dependent on the other equations. This is done by letting `M←(∨/M ≠ 0)/M`.

```
      M                        (∨/M≠0)                M←(∨/M≠0)⌿M
1 ¯3  0  0  4               1 1 1 0              1 ¯3  0  0  4
0  0  1  0 ¯2                                    0  0  1  0 ¯2
0  0  0  1  0                                    0  0  0  1  0
0  0  0  0  0
```

If we now define *D* and *C* as before, `D←+/∧\M=0` and `C←(⍳¯1↑⍴M)∊D` then `C/M` is an indentity matrix with `∧/(+/C)=⍴C/M`. We can, therefore, easily find those elements of a solution which have indices corresponding to the ones in *C* in terms of those having indices corresponding to the zeros in *C*. The number of zeros in *C*, equivalently, the number `K←+/~C` is the dimension of the solution space. In other words, if *B* denotes the general solution we want `K=¯1↑⍴B`. Since we want the columns in *B* to be linearly independent, we can do this by making `∧/∧/((~C)⌿B)=(⍳K)∘.=⍳K`. This can be done by letting `A←(~C)⍀(⍳K)∘.=⍳K`. To finish constructing *B* we need only "fill in" the zero rows of *A* by adding to *A* the matrix `E← -C⍀(¯C)/M`. We illustrate the steps by continuing the example of *M* given above.

```
       M                         D←+/∧\M=0
1 ¯3  0  0  4                    D
0  0  1  0 ¯2             0 2 3
0  0  0  1  0                    C←(⍳¯1↑⍴M)∊D
                                 C
                          1 0 1 1 0


      E←-C⍀(~C)/M
      E
3 ¯4
0  0
0  2
0  0
0  0

      K←+/~C
      K
2

      A←(~C)⍀(⍳K)∘.=⍳K
      A
0 0
1 0
0 0
0 0
0 1
```

The general solution is then `B←A+E`.

```
      B←A+E
      B
3 ¯4
1  0
0  2
0  0
0  1
```

The function, *EASYSOLVE*, can be defined as follows:

```
    ∇ B←EASYSOLVE M;⎕IO;A;C;D;E;K
[1]   ⎕IO←0
[2]   M←(∨/M≠0)⌿M
[3]   D←+/∧\M=0
[4]   C←(⍳¯1↑⍴M)∊D
[5]   K←+/~C
[6]   A←(~C)⍀(⍳K)∘.=⍳K
[7]   E←-C⍀(~C)/M
[8]   B←A+E
    ∇
```

To verify that the columns of `B←EASYSOLVE M` are in fact a basis for the solution space for the system `∧/0=M+.×V` we need to see that the columns of `B` are linearly independent and are all solutions of the system since we know `¯1↑⍴B` is the dimension of the solution space. That the columns are linearly independent is clear because they contain an identity matrix of the proper size. To show that the columns are solutions to the system involves using a rather simple identity which we state in the form of a lemma.

Lemma. Assume `C` is a logical vector, and assume that `M` and `L` are matrices satisfying

```
((⍴C)=¯1↑⍴M)∧(+/C)=1↑⍴L
```

Then,

```
∧/∧/(M+.×C⍀L)=(C/M)+.×L
```

A proof of this lemma is quite easy so is not given. However, informally the lemma is true because the rows of `C⍀L` corresponding to the zeros in `C` are entirely zero and hence the columns of `M` corresponding to these rows make no contribution to `M+.×C⍀L`. The same columns of `M` are eliminated by taking `C/M`. We apply this lemma and recall that `C/M` is now an identity matrix. We will indicate equivalent expressions by writing one of the expressions directly below the other. To show that the columns of `B` are solutions to the system of equations we must show that all the elements of `M+.×B` equal zero.

```
M+.×B

M+.×A+E

M+.×((~C)⍀(⍳K)∘.=⍳K)-C⍀(~C)/M

(M+.×(~C)⍀(⍳K)∘.=⍳K)-M+.×C⍀(~C)/M

((~C)/M)+.×(⍳K)∘.=⍳K)-(C/M)+.×(~C)/M

((~C)/M)-(~C)/M

(~C)/M-M

((1↑⍴M),K)⍴0
```

This together with the other remarks proves that `B` is a general solution of the system `∧/0=M+.×V`.

The other step involved in solving a system of homogeneous equations is to reduce (or change) its matrix of coefficients to an equivalent system of equations, whose matrix of coefficients is in reduced row echelon form. The standard way to do this is by what is called Gauss elimination (not the method used in ⌹). While this method may not be particularly good in most circumstances it is easy to understand and therefore the one we present.

We define a function *REDUCE* that performs elementary row operations to its argument until it is in reduced row echelon form. We must proceed inductively on the columns of the argument *M*. However, we will not proceed inductively (i.e., with a loop) on the rows of *M* since it is unnecessary. (Recall that `⎕IO←0`)

Let `J←0` where *J* will denote the index of the column we are working with. Let `I←(0≠M[;J])⍳1` so that *I* is the index of the first non-zero element in the *J*th column. In order to reduce this column, we want to divide the *I*th row of *M* by `M[I;J]` and subtract from *M* the matrix

```
((I≠⍳1↑⍴M)×M[;J]∘.×M[I;]÷M[I;J]
```

We have now finished reducing the zeroth column of *M* and are ready to proceed to the first column of *M*. An example should show where we are:

```
      M                              J←0
¯2    6    0    1   ¯8               I←(0≠M[;J])⍳1
 0    0    1   ¯3   ¯2               I
¯7   21    0    0  ¯28         0
 2   ¯6    0    0    8


      N←((I≠⍳1↑⍴M)×M[;J])∘.×M[I;]÷M[I;J]

      M←M-N

      M

¯2    6    0    1   ¯8

 0    0    1   ¯3   ¯2

 0    0    0   ¯3.5  0

 0    0    0    1    0
```

At this point we could interchange the *I*th and zeroth rows of *M*. However, we will simply keep track of which row we have used by generating a permutation *P* of the indices which we have used. Thus, we would start with `P←⍳0` and then let `P←P,I`. We also will find it convenient to keep track of the elements `M[I;J]` by letting `Q←⍳0` and each time though let `Q←Q,M[I;J]`.

We are ready to let `J←J+1`. We wish to find the index of the first non-zero element of the *J*th column is not in *P*. We can do this by letting

```
I←((0≠M[;J])∧~(⍳1↑⍴M)∊P)⍳1
```

Notice that this definition of `I` will work equally well when `J←0`. The rest of the steps are the same as before. All we now need are some ending statements. There are two of these necessary. The first is to cover the possibility that for some `J`,`M[;J]` contain only zeros in those places not in `P`. In this case `I=¯1↑⍴M` and we want to move to the next column. The other situation is when `J=¯1↑⍴M` and we are finished.

Rather than defining one function to do it all, we will define a function `REDUCE1` whose argument is a matrix `M` and result is a matrix `R`, the zeroth column of which is the permutation `P`, first column is the vector `Q` and remaining columns form a reduced version of the matrix `M`.

```
      ∇ R←REDUCE1 M;P;J;I;Q;⎕IO
[1]     ⎕IO←0
[2]     P←⍳0
[3]     Q←⍳0
[4]     J←0
[5]     I←((0≠M[;J])∧~(⍳1↑⍴M)∊P)⍳1
[6]     →(I=1↑⍴M)/10
[7]     M←M-((I≠⍳1↑⍴M)×M[;J])∘.×M[I;]÷M[I;J]
[8]     Q←Q,M[I;J]
[9]     P←P,I
[10]    →((¯1↑⍴M)>J←J+1)/5
[11]    P←P,((~(⍳1↑⍴M)∊P)/(⍳1↑⍴M))
[12]    Q←(1↑⍴M)↑Q
[13]    R←P,Q,M[P;]
      ∇
```

The function `REDUCE` is defined by:

```
      ∇ R←REDUCE M;N;O
[1]     R←0 2 ↓M←REDUCE1 M
[2]     N←(∨/P≠0)⌿P
[3]     Q←M[;1]
[4]     Q←(1↑⍴N)↑0
[5]     N←N÷Q∘.×(¯1↑⍴N)⍴1
[6]     R←(⍴R)↑N
      ∇
```

That the function `REDUCE` produces the desired result can be seen by the following example:

```
        M                   REDUCE M
1   0  ¯2   1          1   0  ¯2   1

2   1   3  ¯1          0   1   7  ¯3

4   1  ¯1   1          0   0   0   0
```

Finally, we combine the functions `REDUCE` and `EASYSOLVE` to produce a definition of the function `HOMOSOLVE`.

```
      ∇ Y←HOMOSOLVE M
[1]     Y←EASYSOLVE REDUCE M
      ∇
```

Given a matrix *M*, *HOMOSOLVE M* is a general solution to the system of homogeneous equations. `∧/0=M+.×` . For example:

```
      A←HOMOSOLVE M

          M                   A

1    0   ¯2    1          2   ¯1

2    1    3   ¯1         ¯7    3

4    1   ¯1    1          1    0

                          0    1


      M+.×A

0    0

0    0

0    0
```

The reason for defining the function *REDUCE*1 is that it is useful in a number of contexts. For example, if we wished to find a general solution to a nonhomogeneous system of equations `∧/A = M+.×V`, we can use *REDUCE*1 as follows:

```
     ∇ Y←A NONHOMOSOLVE M;B;B̲;D
[1]    M←REDUCE M,A
[2]    B←,((1↑ρM),¯1)↑M
[3]    M←0 ¯1↓M
[4]    D←+/∧\M=0
[5]    →(~∧/0=(D=¯1↑ρM)/B)NOSOL
[6]    B̲←EASYSOLVE M
[7]    Y←(1↑ρB̲)⌈(ρB)
[8]    Y←((Y,¯1↑ρB̲)↑B̲),Y↑B
[9]    →0
[10] NOSOL:'NO SOLUTION'
     ∇
```

Continuing the example of *M* given above we have:

```
       B←3 1 7 NONHOMOSOLVE M

       B                            M+.×B

 2 ¯1  3                        0  0  3

¯7  3 ¯5                        0  0  1

 1  0  0                        0  0  7

 0  1  0

 3 1 ¯2 NONHOMOSOLVE M
NO SOLUTION
```

If we have defined a function *SIGN* which gives the sign of a permutation, we could define a determinant function as follows:

```
     ∇ Y←SIGN P;N
[1]    Y←×/(,(⍳N)∘.>⍳N)/,(P∘.-P)÷(⍳N)∘.-⍳N←⍴P
     ∇

     ∇ Y←DET M
[1]    M←REDUCE1 M
[2]    →(~=/(⍴M)-0 2)/NOD
[3]    Y←(SIGN M[;0])××/0 0 ⍉ 0 2 ↓M
[4]    →0
[5]   NOD:'ONLY SQUARE MATRICES HAVE A DETERMINANT'
     ∇
```

It should be clear from these examples that the set of functions defined above are essential to a course in linear algebra. We return, therefore, to the original question. Is APL an effective language for communicating this particular mathematical topic? While there may be a number of better ways to use APL in presenting this topic, the fact is that the class that understood the topic was the one that had had one semester of doing mathematics in APL. The other class, for whom the topic had to be presented early on, found an expanded version of this discussion to be so confusing as to virtually prevent them from learning the mathematics. This experience raises serious questions about using APL as the notation for mathematics except when the level of mathematics is equivalent to or below the level of APL sophistication.

Reference

1. England, James W., **Calculus and Linear Algebra in APL**, Amer. Math. Monthly Vol. 85, No. 5 (1978) pp. 371-377.

APL AT SUN LIFE OF CANADA

Stuart Trask
Sun Life of Canada
Montreal, Quebec

1. The Company

Let me first say something about the Sun Life of Canada and its operations, of the volume and nature of our insurance and pension business, so that I may give you an idea of our computer-based administration workload.

We write individual and group life insurance, disability income insurance, annuities, and variable accumulation and other segregated fund policies in Canada, the United States, and Great Britain. We provide service in such areas as investment management, benefit management, and pension fund administration through various subsidiary companies.

At the end of December, 1977 we reported:

Life insurance in force		
Policies		1,539,998
Certificates		2,883,568
Amount - Individual	$15,953,991,187	
- Group	$19,956,590,342	
- Total		$35,910,581,529
Annual premiums - Individual	$ 300,258,469	
- Group	$ 99,954,515	
- Total		$ 400,212,984
Annuities in force		
Contracts		57,619
Annual premiums		$ 59,903,611
Segregated fund policies in force		
Number		26,835
Annual premiums		$ 62,760,360
Health insurance in force		
Policies		8,718
Certificates		1,124,371
Annual premiums		$ 77,366,866
Total Assets		$ 5,542,158,390

Sun Life of Canada is a mutual life insurance company owned by its participating policyholders who are entitled to vote at general meetings of the Company. Dividends are paid to participating policyholders.

Our corporate head office is located in Toronto. Our Canadian administration is centred in Montreal with regional service centres in Montreal, Toronto, Edmonton and Vancouver, and a health claims office in Winnipeg. We have national offices in the United States and in Great Britain. Our U.S. headquarters office is in Boston, and our British administrative office is in London. There are new business offices located in 37 cities in Canada, in 49 cities in the United States, in 29 cities in Great Britain and Ireland, and offices in Bermuda, the Philippines and Hong Kong.

2. Computer Configuration

Our computer configuration relates to this organization. In Montreal we have an IBM System/370-168 with six megabytes of memory. Through a 2860 control unit there are attached thirteen model 8 3420 tape drives, and through four 2880 control units there are twenty-two model 11 3330 disk drives, and sixteen 3350 disk drives, all on line in eight banks.

We have three 3272 control units for the fifty-five 3270 screen terminals located locally in Montreal. The remote screen terminals are attached to the 3705 control unit and there are forty-six in our U.S. headquarters office in Boston, seven in our British administrative office in London, and twenty-seven distributed among our four Canadian regional service centres. These terminals are used for data entry, for servicing record status requests, and for programmer testing. Three of the screens in Montreal and three in Boston are dedicated to our APL system.

Attached also to the 3705 control unit are about six 3780-type remote job entry systems, one hundred and forty-one TWX terminals in our new business offices, the System/370-115 in Boston, the System/3 and a System 3777 in London, and sixteen APL ports. We have about seventeen 2741's in Montreal, three in Boston, and two in London, two Anderson-Jacobson 832's in Montreal, three in Boston, and one in Toronto, all for use with our APL system.

The master scheduler is OS/MVS. We run JES2 release 4.1, and next in priority is the telecommunications access method TCAM message control program. Then we have TSIO as a system task, and APLSV version 3. Then there are the TCAM applications running in about ten initiators, Time Sharing Option (TSO), a subsystem interfacing with TCAM and serving approximately 20 users on average, and finally the batch work - both production and programmer testing and running in about four initiators. We are averaging sixteen thousand jobs per week.

3. APL History at Sun Life

We started using APL in 1970 as soon as a Montreal telephone connection was available from I.P. Sharp Associates Limited. We had a single terminal and three account numbers which we used that summer to develop some pilot projects. In particular we wanted to attract the interest of actuaries. One of my associates, D.R.W. Jamieson, developed a scheme of actuarial notation in APL. This allowed us to present APL as an Actuarial Programming Language and won us an immediate assignment on a critical problem. A project to deliver a rate file for about one hundred and forty British plans was behind schedule, because of specification delays and misunderstandings, and because testing responsibilities were scattered. These table values were generated quite easily in APL with our demonstration package, primarily because the notation used for function names and arguments were the ones familiar to actuaries. With help from a systems analyst to get the numbers out of the workspace and onto tape files the job was done ahead of schedule.

We installed the IBM program product APL\360 in-house for this project, and needless to say APL caught on in our actuarial division. A demonstration package was used for group life and health proposals and actuarial students in these areas were soon writing their own production programs. We adapted an agency manpower planning model from a batch model of the Life Insurance Market Research Association, and we had the attention of Sun Life's market research staff. We prepared a return-on-investment program to demonstrate that it was possible to evaluate all the property investment strategies and cash flows within the time allowed for selection of one. An investment analyst took over and was soon writing his own APL modifications.

In Corporate Planning we were building a corporate model in APL. Our Product Department started doing single premium annuity quotations using APL. Pension proposals were being produced in APL, even pension valuations. 1971 to 1974 was a period of steady and happy growth. Users were learning APL by reading primers, by attending two-day hands-on tutorials, or by just plunging into problem-solving situations with the reference manual handy. In 1974 IBM presented a description of APL at Sun Life of Canada as a case study: "APL Usage in a Commercial Environment".

Then we replaced our MVT operating system with VS2. It was necessary to move our APL users to APLSV. To translate the increased cost to increased benefits we started introducing users to file processing capability provided in TSIO. Files replaced keyboard input for group life rate calculations. Pension valuation data could be read in from OS files. We saved terminal data-entry time but we also went after terminal printing time, directing volume output to our system printers now that this was possible.

Today we have over 800 TSIO datasets on disk. Seventy-five of our one hundred and sixty user account numbers have access to TSIO. APLSV users are also saving at present 1,318 workspaces. For the last nine months we have been providing APL users with an active workspace size of 90K. In the first three months of 1978 we counted 3,466,134 EXCP's for APL, as compared to 565 million EXCP's for all applications in the same period. APL accounted for 45 hours of the total 775 hours of application systems use. The average quarterly CPU utilization for the first quarter of 1978 was 77.1%.

4. Usage of APL

To summarize our APL availability, we provide access in-house to one hundred and sixty user account numbers, through

(a) sixteen dial-up ports for the 2741's and A-J 832's, and

(b) channel addresses to dedicated 3270 screens.

Our Investment Department is the only area of our Montreal operation to use the SHARP APL system currently, primarily to pick up data from the CANSIM data base. Our London office does not have access to our in-house system yet, and their APL work is exclusively SHARP APL. We can share programs and data with our London office through the Sharp system, but I don't believe we have done this more than once.

I'm going to concentrate on our in-house use of APLSV and the TSIO auxiliary processor.

APLSV is available from 0800 hours to 1700 hours Monday to Friday. By request it can also be made available on Saturdays. We do not have sufficient capacity to run APLSV in the evening, and many afternoons we have a demand paging problem which we are gradually overcoming with a variety of changes.

APLSV response time is therefore unpredictable, suffering noticeably in the period from 1430 hours to 1600 hours. Changes are being made in the operating system and each experiment produces some new behavioural patterns in response time, but I think the users are satisfied that improvements are being made.

We started with three screens in Montreal and last winter added a 3286 character printer to provide hard-copy output. Last month we duplicated this arrangement in our Boston office. The screens have proven to be very popular; one user claimed an increase in productivity by a factor of three, with a further doubling when the printer was installed.

I want now to discuss APL use by area. Using the following table to show the extent of use in the first 6 months of 1978, I shall describe some of the work being done.

The Sun Life brief submitted in March, 1978 to the Royal Commission on Financial Management and Accountability, in the section headed Productivity Improvement, contained this paragraph:

> "...The computer is also used by many senior employees and company officers who are skilled in the use of the APL Programming Language and have access to computer terminals. Thus, they are able to devise improved ways of processing data or of developing management information."

I hope my examples will reinforce that statement, but it is also clear, at least to me, that APL is the tool of the end-users for interacting with our forty-odd computer-based administration systems. To quote one such user,

> "APL bridges the gap between traditional computer processing applications and new unorthodox problems previously considered too unstable to warrant analysis through computer processing."

Another user said,

> "When it isn't possible to give a system analyst the lead time he requires to develop new sub-systems, we start with an APL solution which we can refine as we gain experience with the problem. Eventually the APL solution may be replaced by an enhancement to the batch system, but in the meantime we have got a lot of mileage in APL."

APL USE IN THE FIRST 6 MONTHS OF 1978

	Connect Time (hours)	CPU Time (Minutes)	Average No. of Workspaces
Corporate and Canadian			
Accounting	794	501	88
Actuarial	1,067	192	223
Agency Admin. & Research	769	471	82
Group Pensions	687	317	84
Group Life	2,209	639	239
Investment Research	723	438	73
Personnel	34	3	16
Policy Administration	103	19	11
Product	1,229	550	140
Systems	485	98	256
Underwriting Research	535	446	24
	8,635	3,674	1,236
United States			
Accounting	6	1	1
Actuarial	28	25	11
Agency Admin. & Research	3	-	5
Group	193	18	4
Investment	15	1	7
Personnel	16	1	1
Policy Administration	244	491	22
Product	418	182	87
	923	719	138
Total	9,558	4,393	1,374

Accounting

One of the outputs of our accounting system is a set of TSIO files. The data in these files is analyzed in APL workspaces and the usual accounting summary reports prepared and printed. Our budget and cost system includes an end-user report writing facility but many reports were created and printed using APL instead. Our contribution to the LOMA Inter-company Cost Study is put together using APL.

Actuarial

The corporate model is run here now, but other work includes investigation of Income Tax ramifications of actuarial decisions. Interim performance reports are summarized in APL together with forecasts of year-end results.

Agency Admin. and Research

APL is used as the route for interacting with our computer-based agency remuneration system. This department is now developing a complete tie-in between agency production, manpower planning and budgeting. The branch office budgets are assembled and summarized in APL for review by agency management before being directed as input to the corporate batch budget system.

Group Pensions

Some pension plan valuation, in particular the exploration of various assumptions and their impact on total liability. Some pension proposals are also prepared in APL.

Group Life

There is some data collection on the work-load and prospecting activities of field representatives, but the bulk of APL work could be classed as production, the calculation of premium rates for group life, weekly indemnity and long-term disability quotations.

Investment Research

Risk evaluation and bond yield analysis. The department maintains a time series data base covering twenty years on about eighty funds. There is a pension fund portfolio model used to forecast asset allocation for each pension fund. Real Estate analysis to find internal rate of return, and to produce purchase, refinancing, and amortization schedules. This is a production system with fast response requirements, and with occasional special functions being written for an unusual proposal.

Personnel

A salary survey summarized data on a selection of clerical and junior supervisory jobs submitted by a group of Montreal employers. Job descriptions were translated into a grading plan which gives point values for each job. From those point values, and from salary information, some linear regression analysis was done and some graphs plotted in APL by a personnel officer and one assistant. They put together a couple of packages in our public library and produced a 50-page report, 80% of which consisted of graphs and regression analysis printout. The report was distributed to all the participants with very favourable reactions.

Policy Administration

APL is being used to process data collected on the flow of events on one of our products - cheque clearing time, percent of returned cheques, persistency, etc., to evaluate our performance in administering this class of business. Work flow data is also collected on underwriting and policy issue activities to identify peak loads, bottlenecks, and variability in processing time.

Product

Mainly a production system to prepare single premium annuity quotations. This system has been rewritten recently in COBOL to provide automatic turnaround using our TWX network, but APL is still used for about 10% of all quotes, the ones too complex to handle efficiently in COBOL. There is also testing of premium rates for adequacy using asset share methods; development of new insurance plans; dividend calculations, cash values, paid-up values, etc. for special plans; and evaluation of special benefits.

Systems

A skills inventory system is being developed for our Personnel Department, separate from the production personnel system for the usual reasons - uncertain specifications, need for trial solutions - and APL is the only tool that permits this incremental optimization approach. A prototype administration scheme for one type of pension plan is being tested in APL. While APL use in the company is coordinated through the Systems area, corporate programming and systems analysis does not use APL enough, in my opinion.

Underwriting Research

Studies are made of mortality experience. Our company has served as collector of data assembled by many companies on group life experience. The data was summarized and verified and a preliminary report prepared, all in APL, and the same procedure will be repeated next time.

5. Future Growth

Our APL users cover the spectrum from those who execute packages prepared for them to those who are writing those packages, but the majority are people who write APL functions to solve specific problems. We try to provide the sort of environment that encourages the problem solver to use APL. Terminals are in convenient locations, manuals and other instructional material are in plentiful supply, even learner-paced training courses. We have found that ability in APL is not a function of how it is learned but of the urgency of the problem to be solved. We put out a monthly newsletter about system features, new packages, coding techniques, extracts from newsletters of other APL systems, and general information about our configuration of interest to our APL users.

As a result I have seen some unusual file organizations and some strange processing sequences, but so far this has not been costly in the use of computer resources. Our best techniques for control have been to assume that we have responsible users who want to use APL properly and to make use of computer resources in the most efficient way. I think that most commerical organizations using APL have a corporate group of APL programmers similar to a corporate group of COBOL programmers, but we do not. It is my prediction that we will add such a group, but that we will not limit APL use. Our approach has been to encourage any area of the company to use APL if it wishes, and this has unique advantages and disadvantages.

Advantages from the Systems point of view:

(1) Many requests for trivial processing no longer appear on our backlog.

(2) More intelligent user community when discussing system enhancements.

(3) Data processing needs become visible in APL applications, and opportunities can be taken to incorporate that work in a production routine.

Advantages from the users' point of view:

(1) Direct contact between the computer system and the user, no need to interpret the problem to a third party.

(2) Flexibility is retained to modify the solution as the problem is better understood.

(3) Confidence in timely achievement of results and in their accuracy.

(4) Fun to use - problem solvers become highly motivated to use APL.

Disadvantages from the Systems point of view:

(1) Use of computer resources is out of control.

(2) Multiple solutions are being designed to solve similar problems.

(3) Imbalance in use of APL, dependent on user interest in APL.

(4) No operating standards or documentation.

Disadvantages from the users' point of view:

(1) Lack of a batch APL to permit scheduling production.

(2) No assurance of continuity independent from the original programmer.

So I believe we are going to establish a corporate unit of APL programmers. They will provide some general features such as documentation standards for production programs and an ability to run batch APL. They will also search for APL applications in areas of the company not using APL now. Our reference library, our corporate relations department, our system development departments probably have APL applications but lack the knowledge and time to develop APL solutions.

It is true that some things now being done in APL should be included in our existing application systems that are written in COBOL or Assembler. A corporate APL unit can correct these faults although they will not replace the amateur APL user at our installation. It is also true that some programming is now being done in COBOL or Assembler that should be done in APL. Our systems analysts and programmers need to have the capability of assessing such situations and, if the selection is APL, they need the capability to provide the implementation.

There will always be the three classes of APL users at Sun Life,

- Those who run an APL package and provide input.
- Those who use APL to solve particular problems.
- The professional APL programmer who writes APL programs for general use.

We are going to expand our APL capability in this third area.

APL AND THE WHEEL

Neil C. Hershfield
Paterson, Cook Ltd.
Vancouver, British Columbia

Introduction

In this paper, I will discuss the use of APL in actuarial computer applications. I have included a list of some of the applications we have developed in an actuarial consulting practice, but I will not describe any of them in detail. Instead, I will discuss a number of concepts and techniques which we employed in their development. I will also briefly describe the environment of these applications and explain the general design approach we have taken.

The Applications Environment

I was first introduced to APL while working in the actuarial department of a small life insurance company. While I was there, the department found it needed its own access to computing power. Although the computer department ably handled the data processing needs of the company, it could rarely budget the time or personnel to do the cyclical, periodic or special project computing for the actuarial department. We made an organized study of the various languages and time-sharing services then available, and chose APL.

While the representative of the time-sharing vendor was still giving us beginners' lessons in APL, I was avidly skipping ahead in the reference manual and writing our first systems. Right now I blush in embarrassment at the lack of sophistication of these early programs, but, over four years later, some of them are still quite adequate and are still being used.

Systems design and implementation was at first hindered by a lack of time while we had to maintain our regular workload. But gradually as each cyclical chore was systematized and programmed, and each project became a short APL function instead of a multi-columned worksheet, we found time to tackle even larger projects. Thus, even more time was freed. With time available and faster turnaround, there was the chance for more creativity and in-depth thinking.

Currently, the actuary in that company is able to relegate practically all cyclical work to clerks, and uses the time-sharing service as an extremely willing and accurate (if not very clever) actuarial student. Best of all, he does not have to put his thinking down in specifications that can be understood by a non-actuarially trained programmer in the computer department.

At present, I am in charge of computer operations in a quite young, single office actuarial consulting firm. We use the APL service of a well known time-sharing vendor and have one typewriter terminal. We enter small amounts of data directly on our terminal, but have larger amounts keyed onto magnetic tape by an independent keypunching service. We usually have multi-paged reports listed on the highspeed printer. But for speedy turnaround and high quality printing, we do have an almost excessive amount of listings output directly on our terminal.

I am sure that few of you here have had to face exactly the same problems or have been able to enjoy the same freedoms that I have. We had to define and devise a brand new collection of software - from basic data manipulation systems to complex actuarial valuation models. In theory, this all had to be accomplished instantaneously - before the first client project required it. In practice, except for an initial burst of rapid development, it has taken three years. On the other hand, we were fortunate in that we did not have to fit into a pre-existing environment or comply with initial or ongoing standards of a different office.

At the onset, we rigorously laid down a framework for all systems that would be required, and set levels of priority for development. We rejected both outside software we could use on our terminal and other actuarial computing packages that were available to us. We found that practically everything available was too rigid, or emphasized generality in different ways than we wanted. For practical purposes, we did have to start by using some existing packages, but from the beginning, we knew that we would replace them as soon as we could.

We have carefully built up our own library of systems. I would like to say that this was a smooth and controlled process. But of course, it wasn't. We have never had the luxury of time to be completely orderly in our actual development. Although we always had our general framework in mind, segments were often written exactly when and even how they were needed. But always, planned future enhancements or generalizations were allowed for, or partially built into the original implementation. Our level of programming sophistication gradually improved and our use of modularization and utility functions increased, but we managed to maintain each system's particular style. Naturally, as we progressed we had further insights and these were incorporated into the systems. Also, as the APL environment in which we worked improved, we made use of its enhancements.

As haphazard as the development has often been, it has always been under control. Surprisingly little inefficiency and waste has resulted and there has been a minimum of re-thinking and re-working. Even with the firm's expansion and some staff turnover, almost everyone has been involved in design. I can safely say that we have managed to re-invent almost every wheel we have used, and have built up an impressive collection of flexible and general systems of actuarial applications. All the while we have carried on a business and have completed everything in a reasonable time and at reasonable expense.

We have a comprehensive data manipulation system for collecting and carefully editing client employee data, to produce a single versatile data base. To this one base we can apply a host of systems to produce:

- employee statistics
- experience analyses (turnover rates, salary increases)
- updated employee records
- employee benefit statements and subsidiary reports
- group insurance renewal listings
- actuarial valuations of pension plans
- "compensation" calculations for AIB returns.

We also have systems for:

- calculating annuity values on variable and fixed interest and inflation rates
- valuing portfolios of fixed interest securities at different interest rates
- determining estimates of the present values of lost earnings and future support costs, for courtroom testimony
- ranking the performance of pooled investment funds
- performing in-depth studies of historical economic factors through manipulation of raw CANSIM data
- developing pension costs estimates.

As well as our own complete systems, we have systems to augment some of the software packages that we do use:

- PLOT and GRAPHICS
- CANSIM
- CROSSTABS and various statistical techniques
- ACT.

Besides systems for general use, we also maintain actuarial applications of a specialized nature for a number of clients.

Actuarial Computer Applications in General

What has computer technology meant to the actuarial profession? First, what could always be done, can now be done faster. Instead of working through endless multi-columned worksheets, many projects can be programmed and results obtained in minutes rather than days.

Second, problems that could not be solved except by the use of questionable approximations that took too long to complete, can now be done accurately and in reasonable time. I remember in England spending whole days figuring out approximations of complicated multi-life insurances. Now, in minutes, we could present the prospective client with ten options and also produce all the subsidiary cash value and valuation tables as well.

The marginal cost of multiple runs based on different parameters is minimal. Consider also, gain/loss analysis, valuation reconciliations and powerful statistical techniques.

The old standard assumptions and approximations can be thoroughly tested. If necessary, they can be replaced or modified. Or, if they are proven to still be reasonable and adequate, they can be continued to be used with renewed confidence.

Third, many problems and their ramifications that could not even be dared to be considered are not at all formidable now. There is the chance for the use of less crude assumptions, greater accuracy and almost unbelievable complexity. Actuaries can now routinely and easily utilize the following:

- select and ultimate interest and indexing rates
- two-dimensional (age and service related) withdrawal and retirement rates and salary scales
- age dependent proportion-married and spouse-age-difference tables
- secondary decrement (recovery from disability, re-marriage)
- open or closed pension plan membership valuation systems
- cash flow analyses
- complex models embodying interplay between a number of variables.

Actuarial Applications with APL

We have found APL to be a very good language in which to program actuarial applications. In this section, I will discuss a number of what I feel to be the most significant reasons. Certainly, many of the opinions expressed below are equally valid for applications of other disciplines.

APL and actuarial logic in the form of life contingencies blend so perfectly that one can think in both simultaneously. Any translation is practically trivial. A significant amount of actuarial computations involve constructing, manipulating and summing vectors. With APL's conciseness, flexibility and array handling capabilities, formulae are not obfuscated by a myriad of nested loops and programming acrobatics. Consider the elegance of:

```
ANX←+/VT×TPX×PAYMENT
```

`VT`, `TPX` and `PAYMENT` could be variables or functions. This line of code could, by one execution, produce a single annuity factor, a table or even a book. The user could, however, run into workspace-full problems if the APL environment has workspaces of limited size.

Using APL, it is reasonable, and in fact easier, to always start at the basic principles. This is where the methodology and assumptions are chosen, and their ramifications can best be understood. For example, one could challenge the common (m-1)/2m approximation used to obtain non-annual annuity factors. This approximation has been found to be less accurate at the lower ages at today's higher interest rates. Perhaps the solution is to include the next term of this approximation or to develop an entirely new one. Perhaps it would be better to calculate the factor for the actual stream of m payments per year.

Because APL is an interpretive language, functions can be easily edited, or the running of a function can be stopped and re-started with intermittent interactive tinkering. Thus, it is very easy to quickly test new logical enhancements, or run functions for extremely odd and extraordinary cases. Flashes of insight can be tested or put into practice almost immediately. Unlike with most other languages, the costs of desired alterations or improvements are rarely prohibitive.

Other languages require a good deal of study before any actual work can be done. On the contrary, APL can be used as it is learned. A naive programmer may not use the most efficient coding, but he will get his answers. Because APL can be learned and utilized so quickly, actuaries can write their own programs or have their programs written by actuarial technicians. The logical translation is so simple, that there is never the need to translate into an intermediate level of specifications and formulae that can be understood by a non-actuarial programmer. Systems can be up and running immeasurably faster if the programmer understands actuarial principles and notation even if he does not fully understand the particular application itself. The main benefit of this is that the actuary or actuarial department can, with only a small investment in study time, gain access to and control the tool that they themselves want to use. The most difficult interface of all - between the actuarial mind and the programming mind - can be avoided completely or remarkably simplified.

Some Concepts and Techniques

1. Commutation Columns

We have generally rejected programming functions based on commutation columns. As mentioned earlier, APL lends itself perfectly to starting at basic principles where the logic is straightforward and the questions of methodology and assumptions are best faced. More importantly, the use of commutation columns makes it very difficult if not nigh impossible to think in terms of select interest and indexing rates. If one works with vectors of interest and indexing rates, there is absolutely no difference in difficulty in calculating a simple non-increasing constant interest annuity and an annuity where each year the interest and indexing rates are different.

2. Graphs

We have made a good deal of use of plotting facilities. Besides routinely making formal graphs for inclusion in reports, we often use informal plots of function values or historical economic indices to aid in planning and decision-making. Graphs are a quick method to obtain a subjective feel of relationships or effects of changes in parameters. Also, the perusal of a plot of observed values is a good starting point in choosing a smoothing technique.

3. Actuarial Notation

Standard actuarial notation is an extremely concise way of writing down concepts that require a lot of words in English. A minor annoyance has always been that it is difficult in computer reports to make clear but concise column headings or explanatory notes for these concepts. We have all seen and used various methods but found that they are all lacking in various ways. I am currently using a terminal whose functions can be set by the computer. I have written a utility function that, through clever use of super-script, half-index and varying pitch can output quite recognizable actuarial notation. By employing graphic techniques, one could certainly get perfect notation.

4. Variable and Function Names

The actuarial programmer usually approaches the task of developing software from the actuarial, rather than the programming point of view. He will find things far easier if he is able to maintain this stance. A useful aid is always making variable and function names meaningful to the actuarial eye. Our systems are full of names like: YMPE, DOB, FINALAVERAGE, NRA, EECWI and CPP.

5. Interactive Debugging

We have all been repeatedly cautioned to do our thinking and debugging at our desks rather than at the terminal. I disagree. Perhaps I have been lucky in not ever having had to fight for terminal time, and have therefore developed what others would call bad habits. I argue that these are good habits.

Certainly little has to be said in defense of interactive debugging. Besides solving the simplest error of a missing parenthesis, I contend that even major logical problems can best be worked out at the terminal. That is where the problem is awesomely important and when the whole logical structure is fresh in your mind. Also, the active local variables are floating in the workspace and the computer can be used in desk calculator mode for independent parallel calculations. Naturally, I do accept that there is some level of difficulty that makes it more worthwhile to return to your desk.

6. Interactive Running

But more important than interactive debugging, is interactive running. I am not referring to an interactive parameter-setting session but a real partnership between the user and the computer. In actuarial work there are many applications where decisions can be based on intermediate results and alternative logical paths chosen. Also, subsequent runs of models can often be made after only a quick look at the results of a particular parameter combination. We have based our decision on this balance of interactive and desk work on what pattern of action will result in lowest cost to our client. We found quite early on that a slightly higher connect expense was well made up for by a substantially lower personnel charge time. When our time-sharing supplier significantly lowered the connect time charge rate, we found that we adjusted our pattern accordingly.

7. Checking Results

On the other hand, we must realize that a computer is not simply a powerful calculator. Each calculation does not have to be checked, but every possible logical route or odd corner must be. Before a system, or even more importantly, a utility function, can be confidently used over and over, it must be checked by hand calculations or against an independent or already existing function. Some complex actuarial systems we have developed routinely output a listing of methodology, assumptions and approximations. The user then can (and in fact is almost forced to) check that he obtained the answer to the question that he intended to ask. It is also often a good idea to routinely have intermediate results printed out to make it easier for the user to check or duplicate calculations.

8. Utility Functions and Subset Languages

In the development of our software, we have utilized a number of types of subset languages or syntax-readers. APL lends itself naturally to the use of sub-routines for oft-repeated tasks and modularization for clarity or function over-laying.

We have a large library of utility functions. Some are quite standard to any application, such as *CURRENTDATE* or *CENTER*. Others have been written for more specific actuarial needs. The function *TPX* with appropriate arguments of ages and lx's returns a matrix of tpx's. The function *DECDATE* converts a date in DD/MM/YY order into a decimalized form that can be used in calculations. Still others have been written for specific tasks common to our firm. An example of this type of utility function is *PAGETOP* which sets up our standard formatted report quality page heading. Some of these functions are self-sufficient with specific arguments. Others rely on specific globals existing in the workspace. These utlities are used as building blocks in all our major systems and especially in special projects. They are used to simplify all the standardized input and output processes and to handle most conceptual logic blocks in the body of functions.

We have developed a number of subset languages. The actuarial language, ACT, is a good example of this approach. These families of functions can be used much like utilities - as building blocks to build more complicated functions. They can also be used for independent checks when developing a complicated system of programs. They can even be the only functions used throughout an interactive session. An example of this last approach is our `QUANTUM` system of functions that we use to develop courtroom testimony concerning lost earnings and support costs for legal actions arising from personal injuries and fatalities.

State-setting functions are another subset language. In this approach, reasonable values of variables and common or simple logic paths are initially set as default conditions. It requires invoking alternate functions to change the state from these default conditions. We have in general, decided against using this approach because in our applications we prefer to force the user to face each choice and consciously choose an available option.

Because APL has the ability to execute a character vector that is a valid APL expression, we have been able to invent "languages" with such simple syntax and small vocabularies that any user can enter APL code in the form of character input to an interactive program. These user entered lines with the imbedded functions can later be executed in whole, or as a segment of more complicated code. For example, the user could answer the query `'DEFERRED?'`, with `'FOR 10'` or `'TO 65'`.

In a basic data massage system the user can essentially write a specific program of data checks by entering line after line like the following:

```
15 MULTIPLE 5000
3 LESSTHAN COL 6
9 EQUAL 2 3 7
```

The first line would test if column 15 in the data is a multiple of 5,000. The second would test that column 3 is less than column 6. The third would test if column 9 has one of the values 2, 3 or 7.

The most complicated syntax-reader to program, but which in many cases makes user entry the most concise and simple, is where an interactive reply could be in the form:

```
1,2,4/6[2,4],9/11/7
```

The function would first satisfy itself that the syntax was valid and that all numbers and combinations were meaningful. Then the numbers could either determine logical branches within a function or be used to call particular sub-routines or function overlays. The above example could have the meaning:

- retirement benefit formulae 1, 2 and 4
- withdrawal benefit formula - 6 with vested retirement benefit formulae 2 and 4
  - 9 with no vested retirement benefits
- disability benefit formula 11
- death benefit formula 7.

Each of these various subset languages or syntax-readers have their own advantages in specific circumstances. They can be used to easily and quickly write large functions that almost read as English, or they can be used to set up interactive parameter setting functions that are extremely flexible and easy to use. They particularly aid in lessening the problem of APL that coding can be too personal and concise to be understood by other programmers. The more of a function that is written in known sub-functions, the easier it is to read and understand.

A number of these languages already exist in the different APL environments, but they are also reasonably easy to write yourself. Of course, there are pitfalls to avoid when developing your own subset language. Instead of using a safe tried and tested package, you could waste much time duplicating work already done, and end up with a system that is not rigorously tested and documented. But, I contend that unless you can make a very strong case for using an existing subset language, you should write your own. In it, you can embody your own special needs, generalizations, specializations, assumptions and approximations. I am all for solving problems only once, but I say that one should solve one's own problems. One should be able to determine the problem-solving technique as much as possible, rather than have it defined by the available tools.

As I mentioned earlier, ACT is an actuarial APL subset language. I do not feel that I am competent to discuss it here. We have rarely used it other than to occasionally check our own functions. We have also not kept up-to-date with its enhancements. We have sometimes found that compared to our own specific functions, it is not as effective in user time or cost, or is missing an important ability needed for an application. I do know a number of users who have found it to be a very powerful tool.

9. Developing a General System

APL systems can be written modularly and functions can be easily edited. Because of these two capabilities, it is often a good approach to develop a general system by first implementing it as a less general case. In our firm, we have had the need for a lot of quick development, and have thus often had to do this. The system can be designed initially in the eventual all-encompassing general form, but only the parts needed at the time need actually be written. All generalizations should be built in the parameter setting section and allowed for in the coding of the body. This can be achieved by programming branching logic that will not be used or by inserting empty functions. Sometime later when there is time or when it is actually needed, the more complicated case can be relatively simply eased into place. This approach to systems design can also make it easier to add subsequent totally new enhancements, or to gradually build and test an extremely complex model. Remember, an application that works is worth two in the planning stage.

Consider the case where select withdrawal rates are eventually to be used, but aggregate rates are adequate in the early implementation. It would be wise to make withdrawal rates a matrix even though each column will be the same. Later, when the function somehow accepts or constructs two-dimensional withdrawal rates, very few programming changes need be made to actually use them. I could almost recommend in general to make all scalars into vectors and all vectors into matrices.

Conclusions

The world with which actuaries deal is not as simple as it once was. The industry is subjected to increased government regulations and standards. More complex economic and demographic parameters must be utilized. There are pressures for more equitable allocation of costs and distribution of benefits.

New actuarial concepts are constantly being developed. More sophisticated actuarial computer applications are needed to adequately handle many of these new considerations.

We feel that through maintaining clear understanding of our software, we are in a fine position to be able to incorporate any of the industry's or our own innovations. I believe that this necessary understanding has been achieved by developing our systems on our own, from first principles.

ADMINISTRATION APPLICATIONS FOR APL

David J. Thomson
The Equitable Life Insurance Company of Canada
Waterloo, Ontario

The life insurance industry is experiencing increasingly competitive conditions. In this environment, it is crucial that management carefully review and evaluate alternative operating procedures in order to improve the overall competitive and profitability positions of their company. Because the life insurance industry is labour intensive, alternative operating procedures often involve full or partial computerization in an attempt to improve operating leverage by shifting from variable costs per unit transaction associated with manual processing, to a primarily fixed system cost under fully computerized activities. With this increasing emphasis on computerization, management must take care to examine all possible alternatives with respect to both the ultimate operating system being evaluated, and the methodology employed in the development process. In each of these aspects, APL warrants greater consideration than it has been given in the past.

The main points to be covered in this paper are:

1. The identification of examples of uses of APL outside the Actuarial sphere.
2. Clarification of some misconceptions which have tended to block the use of APL in the past.
3. To review the consequences of running APL in-house, in terms of our experience at Equitable Life.
4. And finally, to take a brief look at what the future holds in terms of administrative applications for APL in the insurance industry.

The use of APL in the insurance industry has often been considered the exclusive domain of the Actuarial wizards. Historically, only the mystic powers of the Actuary and his staff were potent enough to procure access to external time-sharing systems. An analogy can be drawn to the days when only the king and his legion of honour rode horses, while the masses walked in the aftermath. It is hoped that this power is now being shared somewhat more liberally with other sectors of life insurance management.

APL can be of use to a wide range of functional areas in an insurance company; administration, finance, personnel, and marketing, to name but a few. APL is supportive to the management functions of studying costs, work flows, operating systems modelling, and for production systems - although I'm sure there are a number of EDP Officers who would choke on the latter of these.

If the responsibilities of line area management can be summarized as planning, implementing, doing, and monitoring the functional processes for which they are responsible, then some of the potential applications for APL can be discussed in this framework.

With respect to planning, APL is widely used for model building and forecasting. For functional systems being implemented in a computerized form, APL can be used both as a tool for designing computer systems, and in some instances is appropriate as the language in which the ongoing system can be written. Also, during computer system implementation, APL can be used as a method of supporting data capture and validation prior to system implementation. From the standpoint of controlling operations, APL can be used to prepare adaptive and timely management reporting procedures.

In order to provide a few illustrative examples of the types of uses of APL, let us turn to the major functional areas within an insurance company.

In the investment area, three examples come to mind. The use of APL in conjunction with "linear programming" techniques for optimization can be used to improve aggregate rates of return on investment. In this situation, forecast returns for various investment media such as bonds, mortgages, various groupings of stocks, treasury notes, etc., and risk premiums associated with each investment media, can be combined through the linear programming procedures to obtain a maximized investment return within the constraints imposed by legislative restrictions on investment mix. A second example from the investment area is the use of the TSE or Financial Post data bases to make fund performance evaluations and comparisons. Another example is the evaluation of discounted cash flows to determine the post-tax effects of bond trading under changing market conditions.

In the financial planning area of the company, APL can be used for forecasting cash flows and in turn to assist in short term cash management and investment. A second example from the finance area is the use of APL in the evaluation of complex buy versus lease comparisons using discounted cash flow techniques.

In the marketing area, APL can be used for applications ranging from performing intercompany rate comparisons to customized estate planning analysis. A specific application might be the evaluation of new product profitability. Using Monte Carlo simulation methods one can evaluate the effects of market size, market share, development cost, experienced mortality, administrative expenses, and sales expenses under various assumptions as to the averages and standard deviations for probability distributions associated with each of these factors. This methodology has been discussed in a number of books such as "New Power for Management" by David Hertz. Using this technique, projected cost-volume-profit analysis can be derived and so provide quantified information as to the likely profitability profiles of alternative new products.

An example from the office services area is the use of APL-based software packages which are available for the manipulation of data for phototypesetting. In terms of the preparation of rate books in the insurance industry, this facility is particularly useful when the rates have been calculated using APL. Significant savings in manpower and time can be obtained through the reduction in clerical time required for proofreading and error correction.

In the area of administration of individual and group insurance contracts, a number of applications are feasible. APL may be used to expediently develop interim administrative capabilities, while you are waiting for your computer experts to get around to responding to your request for system revisions to support new products. This is beneficial in that it affords the opportunity for administrative staff to identify the majority of the quirks designed into the product by the actuaries before the EDP staff have the opportunity to compound them when revising existing computer systems to accommodate the product. By initially processing new products through an APL system, major expenditures can be deferred until the product has proven itself in the marketplace and characteristics stabilized. A side benefit is that by having the data in a machine-readable form, the conversion effort which would be required had the product been manually administered initially is reduced.

A second example of administrative applications for APL, which is being used at Equitable Life, involves the preparation of APL programs to collect data to be loaded into large scale data bases. In the specific situation where this technique was used recently at Equitable Life, some information which had not previously been a machine processable form was required for 22,000 of the lives which were to be included in a data base. In order to permit clerical staff which was familiar with the data involved to transfer this information into a processable form, and avoid keypunching, several short APL programs were prepared. The output from the procedure was an APL file which could be loaded directly into the data base which was under construction during the three-month period in which the data was being captured.

This procedure permitted us to avoid hiring additional staff or using overtime, and yet carefully validate the data being transformed.

A further example where APL has been used in our group administration area is the processing of cost study data for specialized work flow and transaction time analysis.

Varily, even in the sanctitude of the EDP area, practical applications for APL can be identified. Ignoring the obvious use of APL for production systems, (as EDPers are prone to do), APL can be effectively employed as a method of developing and maintaining on-line documentation, generating test data, pre-testing system logic, and as a tool in the system design work.

Based on the foregoing examples, we can see that APL can be realistically considered a functional management tool, beyond actuarial calculations.

Turning now to the area of common misconceptions, or at least unproven folklore -- I have heard the phrases "drains the system", "highly inefficient", and "CPU hog" too often! (Usually from people who neither understand nor have used APL.) After considerable research, I have been unable to find any empirical data which supports the belief that APL is run-time inefficient. This is not to say that APL is as CPU efficient as COBOL or PL/1 or FORTRAN, but rather that there does not appear to have been any scientific research applied to this topic of relative efficiency. There just isn't any factual information on the subject.

Another common misconception is that APL is ineffective when dealing with large data files. Evidence of the inaccuracy of this can be seen by referring to the presence of large scale data bases available on the I.P. Sharp system, and in the variety of data file structures which are supported by smaller scale APL systems, such as APLSF on the DEC system 20 computers.

One of the most common misconceptions about APL is that the APL user must be a mathematical genius (or a fulltime programmer). At Equitable Life, we recently trained four clerical staff members in my department to use APL for simple administrative calculations which did not justify the development of formal computer programs. This training took place in four 1 hour sessions, with clerical staff which had no prior experience with computer programming. The people who were trained have been successful in applying APL in their normal day to day work.

Equitable Life has recently migrated most of the APL applications which had previously been running on a time-sharing system onto our own in-house computer. The following benefits have been identified:

1. It is much easier for us to access data which is available from corporate files, since the transfer of data is now completely internal to a single computer configuration, rather than requiring tapes to be produced and shipped to the time-sharing service's site. This has improved our ability to obtain timely information.

2. In addition, improved information flow is available in the situation where substantial amounts of printed output are generated, since highspeed printer output merely moves within the building rather than being shipped from the location of the service bureau to our offices via courier or mail.

3. The transfer pricing scheme being used for computer usage at Equitable is sufficiently lower than the charges which we were incurring through the time-sharing service, so as to permit the justification of programming procedures which under the former cost structure could not be cost justified.

Even though significant benefits have been derived as a consequence of our in-house migration, this should not be considered a panacea. We continue to use service bureau time for access to the Financial Post data base and various other software packages, which are too complex to develop or maintain ourselves. If your company is considering in-house migration, careful evaluation must be given to the advantages of access to the network facilities supported by time-sharing services, and the technical and training assistance offered by service firms.

Looking to the future I think it reasonable to expect that greater emphasis will be placed on non-actuarial applications for APL in the insurance industry, primarily in terms of its use in supporting management science techniques such as those I discussed earlier. The potential use of APL for ongoing systems is less clear. For this type of application to be viable in a disciplined environment, it will be necessary for some preliminary evaluation to be done to quantify the relative efficiency of APL in terms of:

(i) CPU requirements
(ii) file storage
(iii) manpower requirements for system development and maintenance

as compared to alternative high level languages available at a given company. This type of quantification should permit more effective decision making when evaluating alternative methods of system construction, and as well, shed some factual light on a subject which is encrusted in folklore.

The use of APL to support management science techniques and ongoing systems will not be breaking new ground so far as APL is concerned, since I believe this type of application is becoming fairly widespread in the business community. However, much of this ground is yet to be covered in our industry, and I believe this presents a challenge to us all.

DRAFTING WITH COMPUTER AIDED GRAPHICS

Bernie P. Atutis
The Raymond Corporation
Greene, New York

Introduction

For nearly a decade, The Raymond Corporation has been developing and improving the Narrow Aisle concept. Raymond manufactures battery powered lift trucks, both in the United States and Canada, and distributes them through its dealer network.

In its endeavour to provide high quality Narrow Aisle lift truck applications, Raymond has engineered a wire guidance tracking system. Raymond System Engineers, working through the dealerships, design and install wire guidance systems in manufacturing and warehousing facilities. This wire guidance grid system is usually installed in the center of a defined traffic aisle and about 4 centimeters beneath the floor's surface on which Raymond battery powered vehicles travel.

Raymond has many types of guidewire systems for various Raymond Narrow Aisle material handling vehicles, from a simple guidance for vehicle steering in aisles, to a complex system of vehicle guidance, positioning and vehicle operator communications.

Raymond also has engineered an intelligent floor in which its horizontal transport non-man aboard "Electote" vehicle travels. This intelligent guidewire system both positions and guides the Electote vehicle within the facility.

The demand for Raymond systems is ever-growing with several hundred wire guidance systems installed and nearly 100 additional floors in design pending installation. This year Raymond Management decided to use a computer-aided graphic plotter to draw the blueprints for the several drawings it needs to install each guidewire system. This technical paper presents the technique used by the Raymond "Narrow Aisle People" to produce the needed drawings.

In the most complex system, there are as many as six wires embedded in the floor, carrying five different frequencies: Main Aisle, Pick Aisle, Right Turn, Left Turn, and Wire Cross. In addition, vehicle communications provide position data and operator information between the vehicles and the computer. This information is processed on frequencies carried on the Main and Pick wires within the guidance system. A typical 25 aisle guidepath for a computer controlled system has an estimated one mile of floor cut and four miles of guidance wire.

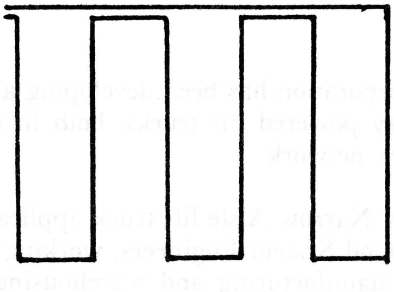
Typical Computer Controlled Guidance Data

An Aisle Code identifies each aisle of the warehouse and is used to initialize any vehicle assigned to the minicomputer's guidepath. It locates a computer controlled vehicle and begins to monitor its movement as it travels on the guidepath. Each increment of travel is monitored by the vehicle, and the vehicle status table data within the computer is compared to vehicle monitor system for positioning the selected vehicle within the warehouse.

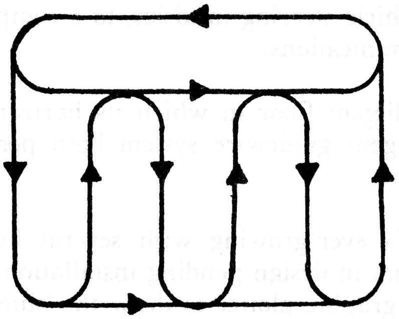
Typical Guidepath Design

There are four basic designs with a multiple of options that can be configured to match a specific warehouse.

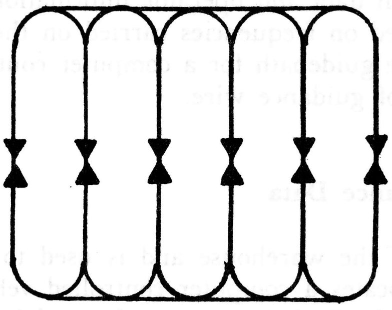
"A" Type Bi-directional and provides narrow aisle vehicles a guidepath within pick aisles.



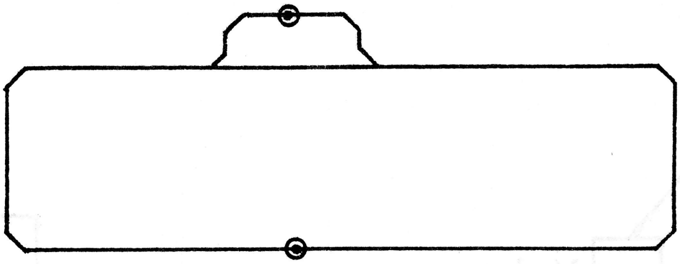
"B" Type Provides simple guidepath for narrow aisle vehicles traveling in a single direction. Option of being installed as an upgradeable floor.



"C" Type Bi-directional travel and provides communications for computer-controlled vehicles. Other narrow aisle vehicles can use the guidepath for steering within the pick aisles.

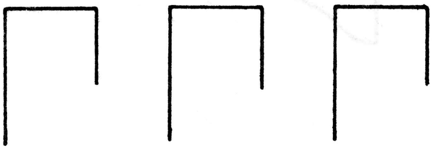


"D" Type Provides an "intelligent" guidepath for the Electote vehicles.



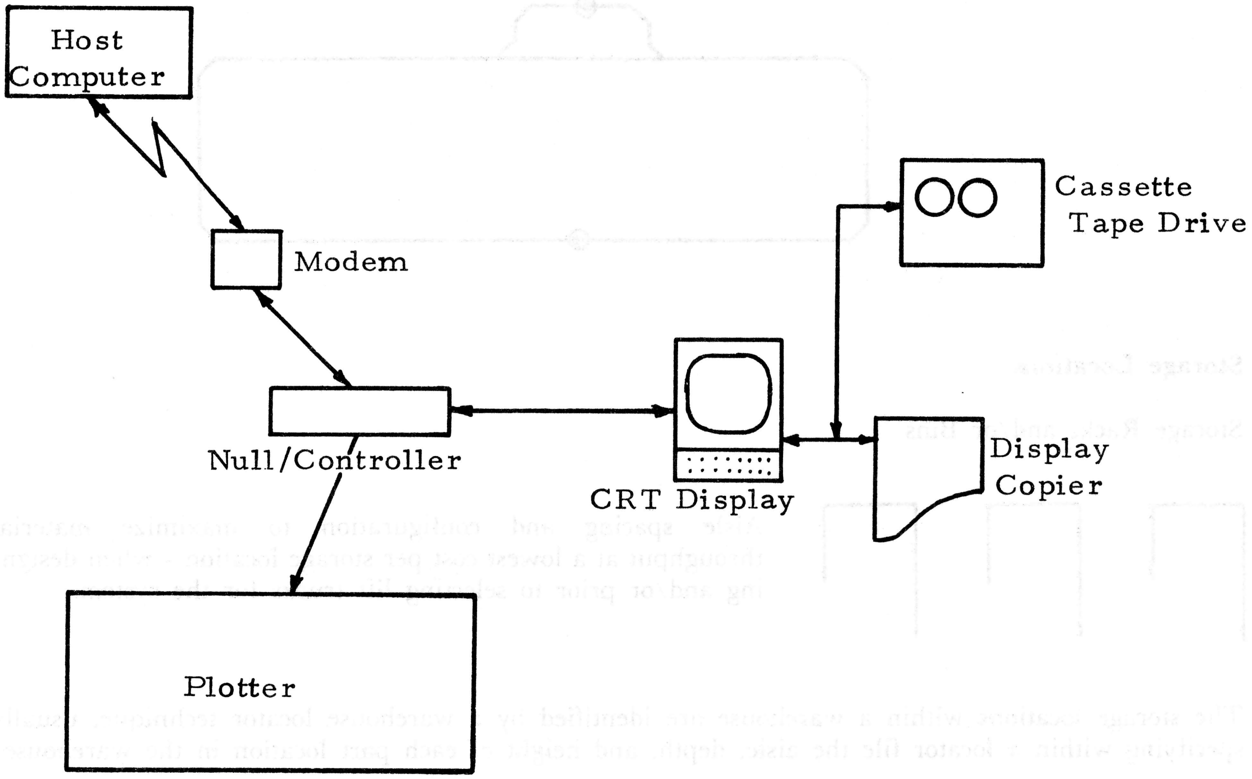
Storage Locations

Storage Racks and/or Bins



Aisle spacing and configuration to maximize material throughput at a lowest cost per storage location - when designing and/or prior to selecting lift trucks for the system.

The storage locations within a warehouse are identified by a warehouse locator technique, usually specifying within a locator file the aisle, depth, and height of each part location in the warehouse.



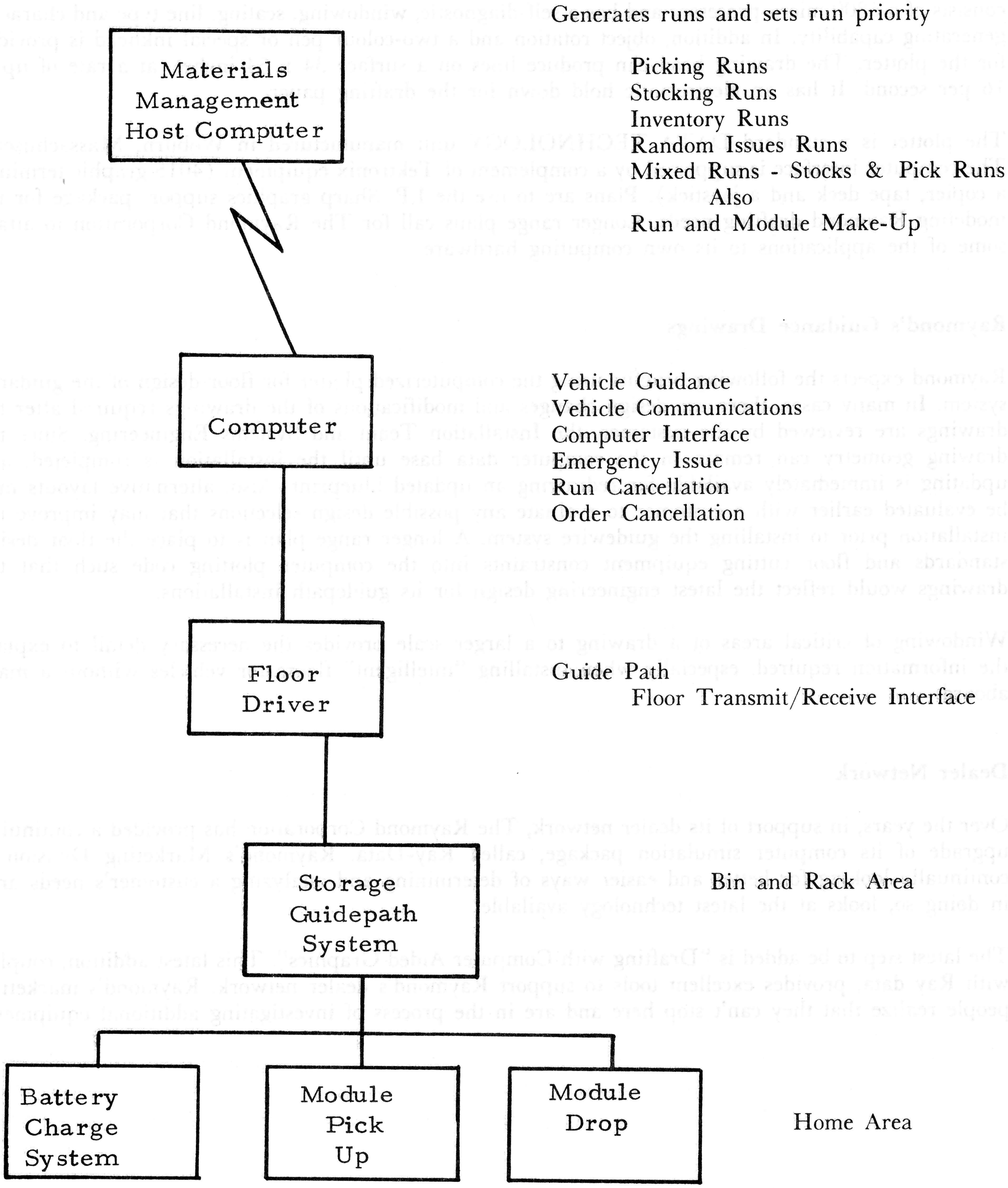
RAY-DATA DRAFTING PLOTTER OVERVIEW

Each storage location in the pick Aisles are identified as being on the left or right, depending on travel direction 1 or 2, such that the vehicle direction is known. The computer determines if either the right or left bin light must turn on to assist the vehicle operator in locating the material specified on the on-board video display. After performing the specified task, the Task Complete button is depressed, thus signalling the computer to position the vehicle to the next warehouse location specified in that vehicle's assigned run.

The Depth traveled in any aisle depends upon the aisle direction being traveled, 1 or 2. A simple logic routine either subtracts or adds each 2-inch increment traveled within the aisle, always looking for a null position within the aisle where action is required.

The Heights are selected by a table within the mini-computer. Alpha characters are assigned to any multiple of 6 inches; i.e., "C" = 18 inches. Warehouse banding technique is used for storing material. Higher moving material is usually kept at lower levels to minimize lifting the operator and module.

Computer Guidance and Control Data Flow


Generates runs and sets run priority
Materials Management Host Computer
Picking Runs
Stocking Runs
Inventory Runs
Random Issues Runs
Mixed Runs - Stocks & Pick Runs
Also
Run and Module Make-Up
Computer
Vehicle Guidance
Vehicle Communications
Computer Interface
Emergency Issue
Run Cancellation
Order Cancellation
Floor Driver
Guide Path
Floor Transmit/Receive Interface
Storage Guidepath System
Bin and Rack Area
Battery Charge System
Module Pick Up
Module Drop
Home Area

Computer Drafting Hardware

The computer aided drafting table responds to commands built into its own firm-ware. This firm-ware consists of a 6800 micro-processor and has a self-diagnostic, windowing, scaling, line type and character generating capability. In addition, object rotation and a two-colour pen or special inkhead is provided for the plotter. The drawing head can produce lines on a surface 34 x 54 inches, at a rate of up to 16 per second. It has an electrostatic hold down for the drafting paper.

The plotter is a standard DATA TECHNOLOGY unit manufactured in Woburn, Massachusetts. The computer interface is supported by a complement of Tektronix equipment (4015 graphic terminal, a copier, tape deck and a joystick). Plans are to use the I.P. Sharp graphics support package for the modeling Raymond drafting needs. Longer range plans call for The Raymond Corporation to attach some of the applications to its own computing hardware.

Raymond's Guidance Drawings

Raymond expects the following benefits using the computerized plotter for floor design of the guidance system. In many cases, there are design changes and modifications of the drawings required after the drawings are reviewed by the customer, the Installation Team and Systems Engineering. Since the drawing geometry can remain in the computer data base until the installation is completed, any updating is immediately available for redrawing an updated blueprint. Also, alternative layouts may be evaluated earlier with a customer to evaluate any possible design selections that may improve the installation prior to installing the guidewire system. A longer range plan is to place the floor design standards and floor cutting equipment constraints into the computer plotting code such that the drawings would reflect the latest engineering design for its guidepath installations.

Windowing of critical areas of a drawing to a larger scale provides the necessary detail to explain the information required, especially when installing "intelligent" floors for vehicles without a man-aboard.

Dealer Network

Over the years, in support of its dealer network, The Raymond Corporation has provided a continuing upgrade of its computer simulation package, called Ray-Data. Raymond's Marketing Division is continually looking for better and easier ways of determining and analyzing a customer's needs and, in doing so, looks at the latest technology available.

The latest step to be added is "Drafting with Computer Aided Graphics". This latest addition, coupled with Ray data, provides excellent tools to support Raymond's dealer network. Raymond's marketing people realize that they can't stop here and are in the process of investigating additional equipment.

APL GRAPHICS FOR TREND ANALYSIS AND PLANNING

Vicki Hill
Steak and Ale Restaurants of America, Inc.
Dallas, Texas

Introduction

This is the story of a restaurant chain's experiences in developing a system of computer graphics for analytical and planning purposes. After meeting early defeats with another system, we turned to I.P. Sharp for a solution to our problems. The system they designed for us is very popular with our executives and can be maintained and utilized by nontechnical personnel. We are **users**, not programmers, and we are using the power and flexibility of APL to solve some very practical business problems.

Steak and Ale is a limited menu restaurant chain specializing in steaks and seafood. We have 158 restaurants in 31 states and 2 restaurants, known as Three Crowns, in Toronto. Most of our restaurants in the United States use the names Steak and Ale, Jolly Ox, or Bennigan's. Since our first restaurant opened just 12 years ago, one of the big challenges we face is managing the rapid growth that our company has experienced.

In 1976 we turned to graphics as an aid in two problem areas. First, with over 100 restaurants, we needed some means of tracking the performance of individual restaurants. Our in-house computer provided accounting data, but all planning and analysis was done either manually or through outside time-sharing services. Consequently, if an executive needed an analysis of a particular restaurant, we would gather data from our accounting reports and graph it by hand.

Secondly, we were increasing our emphasis on marketing strategies in 1976. Since this required a look at all of our restaurants in a given market, we needed the ability to analyze them as a cluster of restaurants. This required summing the sales of all restaurants in the market to get the total sales for the market cluster before analyzing the market cluster. Again, this was done manually. The question arose: "Can the computer help us out?"

Our First Fiasco

Not fully realizing the complexity of computer graphics, we explained our problems to a time-sharing service's representative and asked for a solution. He agreed to develop a FORTRAN program for us which would access a data base of sales by restaurant and plot moving averages of sales for individual restaurants or for clusters of restaurants. The only costs to us were to be the computer time for development and the costs of the graphs, which he estimated to be $1.50 each. Since we planned to run about 200 graphs per month, we were expecting bills of $300 per month and no major problems.

The system never met our expectations. The biggest problem was that the time-sharing service did not have access to a local plotter. Our graphs were mailed to us from another state and usually arrived about a week after the job had been run. The delay would have been bearable if the graphs had been correct. Unfortunately, time after time we would receive 200 graphs which were unusable due to a program error. Since the local programmer could not see the results of program changes until the graphs arrived in Dallas, we often had to rerun graphs 2 or 3 times.

When we finally received good graphs, we had to make copies for all of the executives. This was a 3 day clerical chore, since the graphs came on 200 foot rolls. Each graph was 11″ by 17″. They had to be cut apart and individually reduced. Next they had to be arranged in a different order, since we had no control over the order in which they were plotted. Finally, sets were reproduced for all of the executives.

The graphs themselves were an instant hit with the executives. They could see trends and seasonal patterns more clearly than with any earlier types of analysis. Consequently, they wanted more graphs with additional data bases and expanded graphic capabilities. However, to the people involved in producing the graphs, the system was a giant headache.

We knew the development work could be expensive, so we were not too surprised when our first month's bill was $3,000. However, we never saw the $300 bills which we had expected. Every month the bill was between $2,000 and $3,000. The representative kept showing us how an individual graph cost $1.50, but our bills were still over $2,000 because of problems with the system. In fact, the system was never reliable enough that we could request graphs or change data without contacting the representative to do it for us.

After 8 months, we admitted defeat. We decided to start over, but this time we would thoroughly investigate available systems and learn from our mistakes.

The Second Time -- Success

Our first concern was a convenient plotter, preferably in-house. Due to the cost of graphic software and hardware, our systems department advised us to continue to use outside time-sharing for this application. We talked with several time-sharing services which offered graphs for $4 to $15 each compared with the $1.50 we were supposed to pay whenever the problems with our current system were worked out. We were almost ready to lease a plotter and accept the offer of $4 graphs when we got a phone call from the Dallas office of I.P. Sharp.

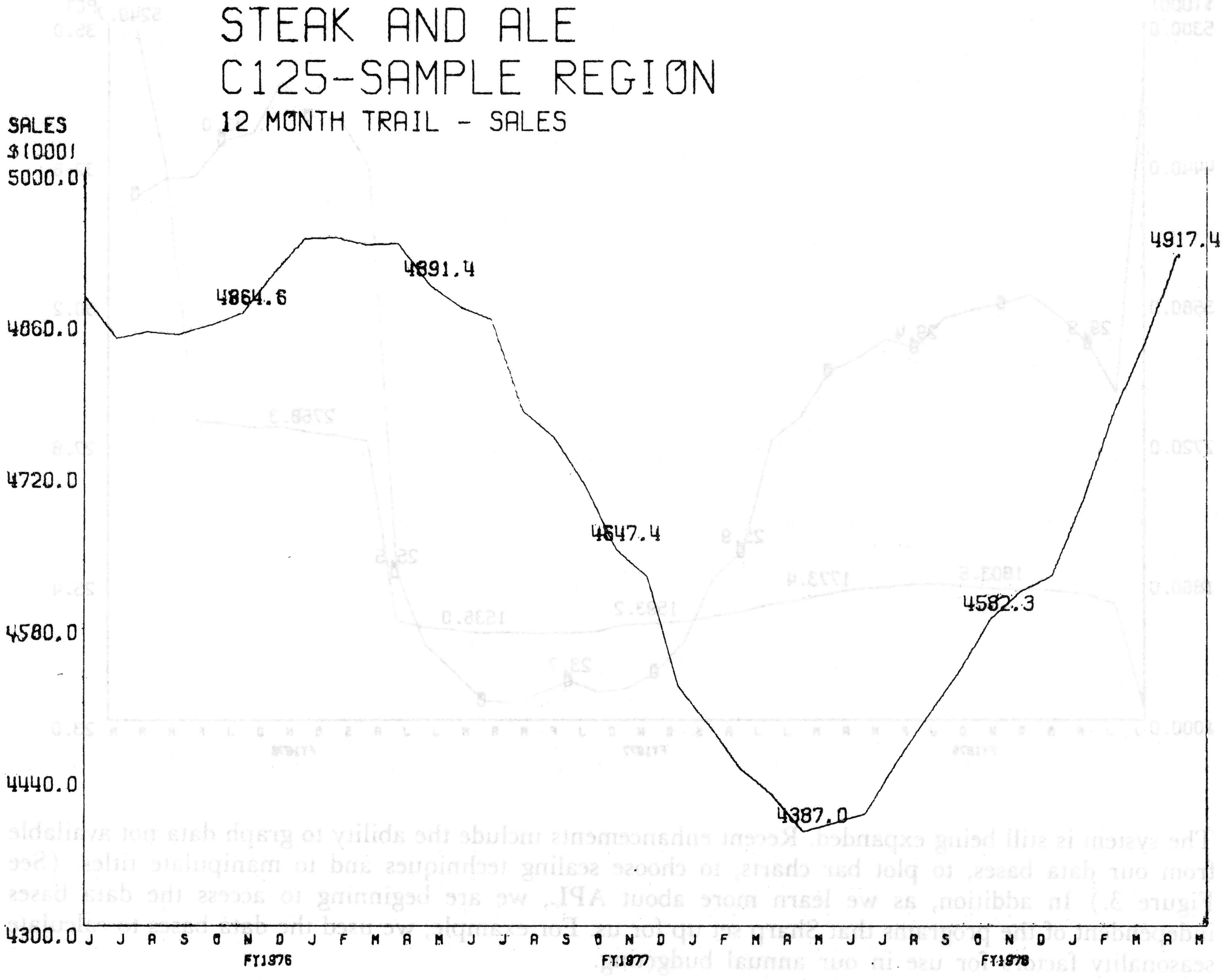
At our first meeting, we explained our requirements to the Sharp representatives. We wanted an in-house plotter which would take 8 1/2″ by 11″ fan-fold paper to avoid our reproduction problems. We wanted a simple 1 line graph that would be inexpensive to produce in large quantities. We wanted to be able to request graphs ourselves and to print them out within 24 hours of the request. Finally we wanted a program that would be adaptable to future enhancements. The Sharp people just smiled, told us our requests would cause no problems and asked for an opportunity to give us a demonstration 5 days later. We agreed, although we were still not optimistic that they could meet our requirements.

Five days later they were back with a plotter. To our amazement, they produced the graph that had caused us months of problems with the other services. I asked to see the program that had produced the graph. The main function had 22 lines of code. The FORTRAN program which we had been using had over 500 lines of code. In addition, the Sharp graphs cost no more than the ones we had been getting. We were all impressed, but I still was not sold on APL and the final decision was mine. The Sharp people invited me to an upcoming class to learn APL for myself. By the second day of class I was a convert to the ease and efficiency of APL programming. We signed a contract with Sharp and a month later our new graphics system was operational.

Our Graphics/Trail System

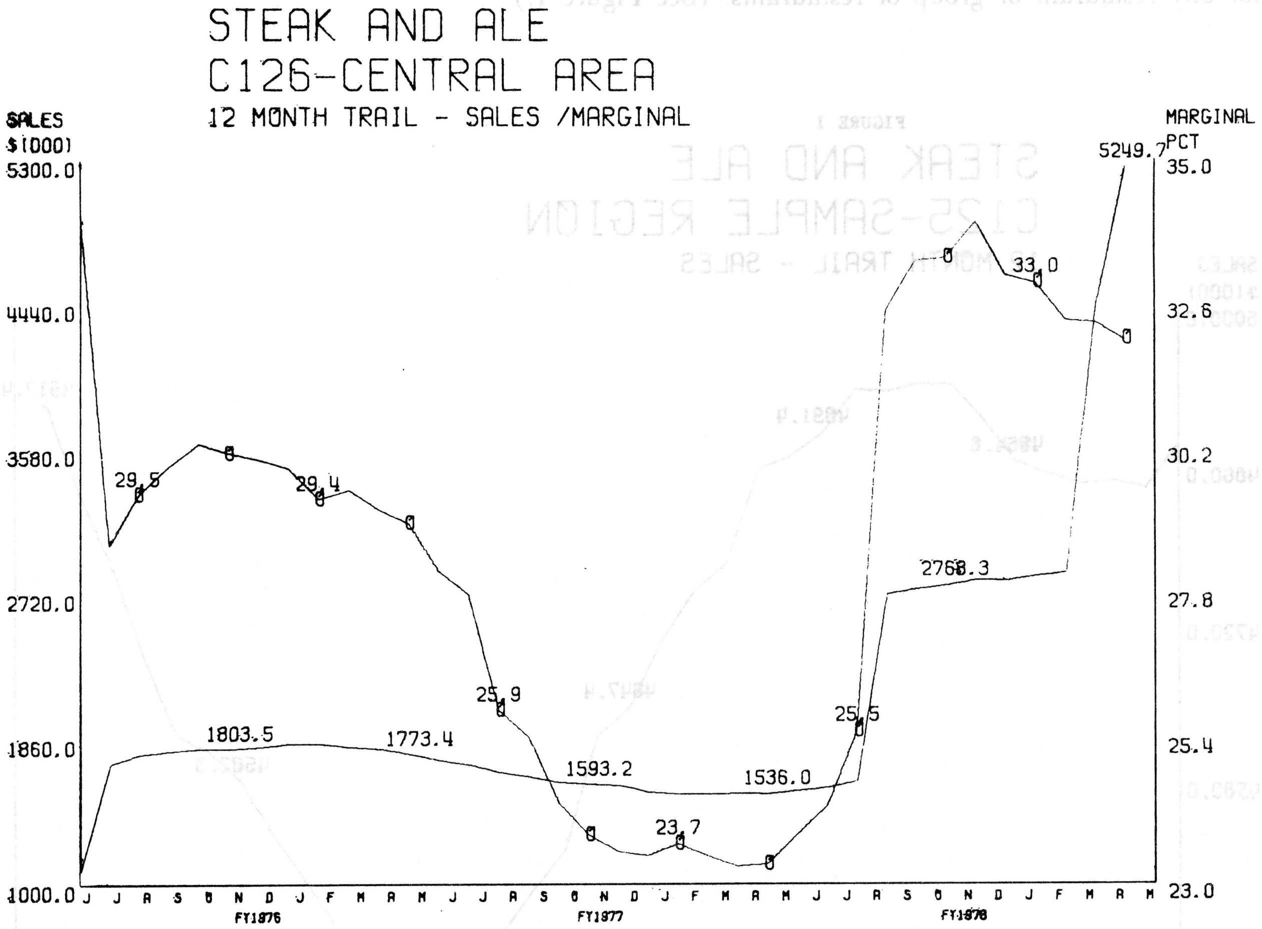
Initially we set up one data base with a six year history of sales by restaurant by month. We liked to examine trends by graphing moving totals of sales figures. For example, if every data point consists of the latest 3 months of sales, then we have a 3-month trail of sales. This helps to eliminate seasonal fluctuations for analytical purposes. Initially our system was set up to permit us to plot moving trails for any restaurant or group of restaurants. (See Figure 1.)

FIGURE 1



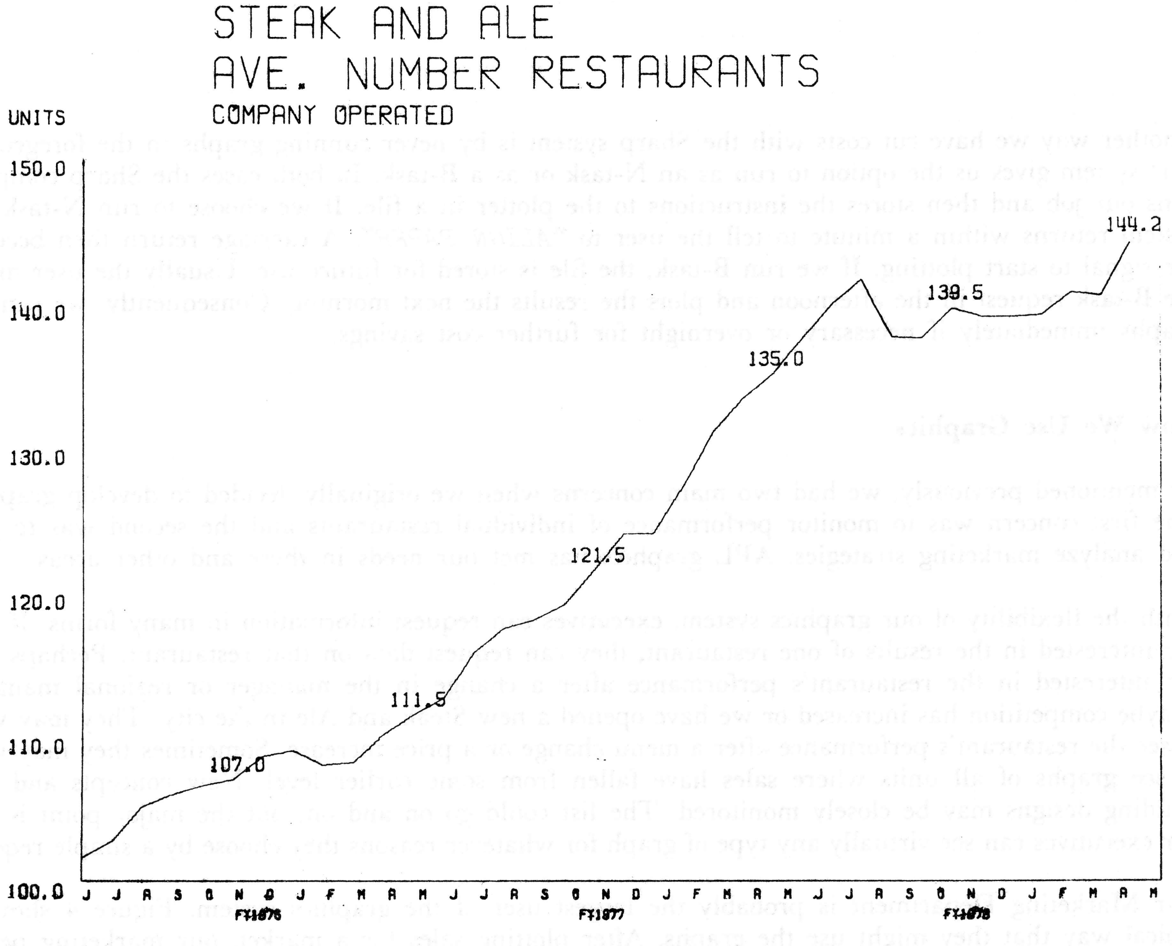
As time progressed, our executives requested enhancements to the system. Thus far we have added data bases for three levels of profit measurement, for budget data and for statistical information such as price indices. We can now plot 1 line or 2 line graphs and we can request data in actual dollars, in constant price-level dollars, as a percent of sales, etc. (See Figure 2.) In fact, now our executives have a difficult time deciding how they want the data displayed.

FIGURE 2



The system is still being expanded. Recent enhancements include the ability to graph data not available from our data bases, to plot bar charts, to choose scaling techniques and to manipulate titles. (See Figure 3.) In addition, as we learn more about APL, we are beginning to access the data bases independent of the programs that Sharp set up for us. For example, we used the data bases to calculate seasonality factors for use in our annual budgeting.

FIGURE 3



Plotting With Sharp Graphics

Steak and Ale's graphics system uses Sharp workspace 3 *GRAPHICS*. One of the key advantages to the functions in this workspace is that they are primarily terminal independent. That is, the programmer can decide what he wants to do, set it up, and then decide what type of device to use for plotting. The functions are designed so that when the programmer chooses the terminal, only one function, called *OUTTERM*, is dependent on the terminal chosen. These characteristics of 3 *GRAPHICS* enabled the Sharp representatives to set up our demonstration so rapidly, even though they had little experience with the type of plotter we required.

A second advantage to the terminal independence of 3 *GRAPHICS* is that we can easily change plotters as our needs or technology changes. Currently we are using a Houston Instruments DP-1 plotter. If a plotter that more closely meets our volume or cost requirements comes on the market, we are assured that our software could be easily changed to use the new device. With rapidly changing technology, this flexibility is important to us.

Due to the volume of graphs we needed, cost was an extremely important consideration. By careful use of 3 *GRAPHICS*, the Sharp representatives have kept our cost per graph low. Workspace 3 *GRAPHICS* contains two basic functions: *DRAW* to draw lines and *TEXT* to draw characters. Each of these functions uses a matrix of control data and X-Y coordinates. By designing the system to first draw all lines and then draw all characters, we make only one call to *DRAW* and one call to *TEXT*. This saves considerable initialization costs. We are still looking for a way to save further by eliminating the separate call to *TEXT* entirely.

Another way we have cut costs with the Sharp system is by never running graphs in the foreground. Our system gives us the option to run as an N-task or as a B-task. In both cases the Sharp computer runs our job and then stores the instructions to the plotter in a file. If we choose to run N-task, the system returns within a minute to tell the user to "`ALIGN PAPER`". A carriage return then becomes the signal to start plotting. If we run B-task, the file is stored for future use. Usually the user makes the B-task request in the afternoon and plots the results the next morning. Consequently, we can run graphs immediately if necessary or overnight for further cost savings.

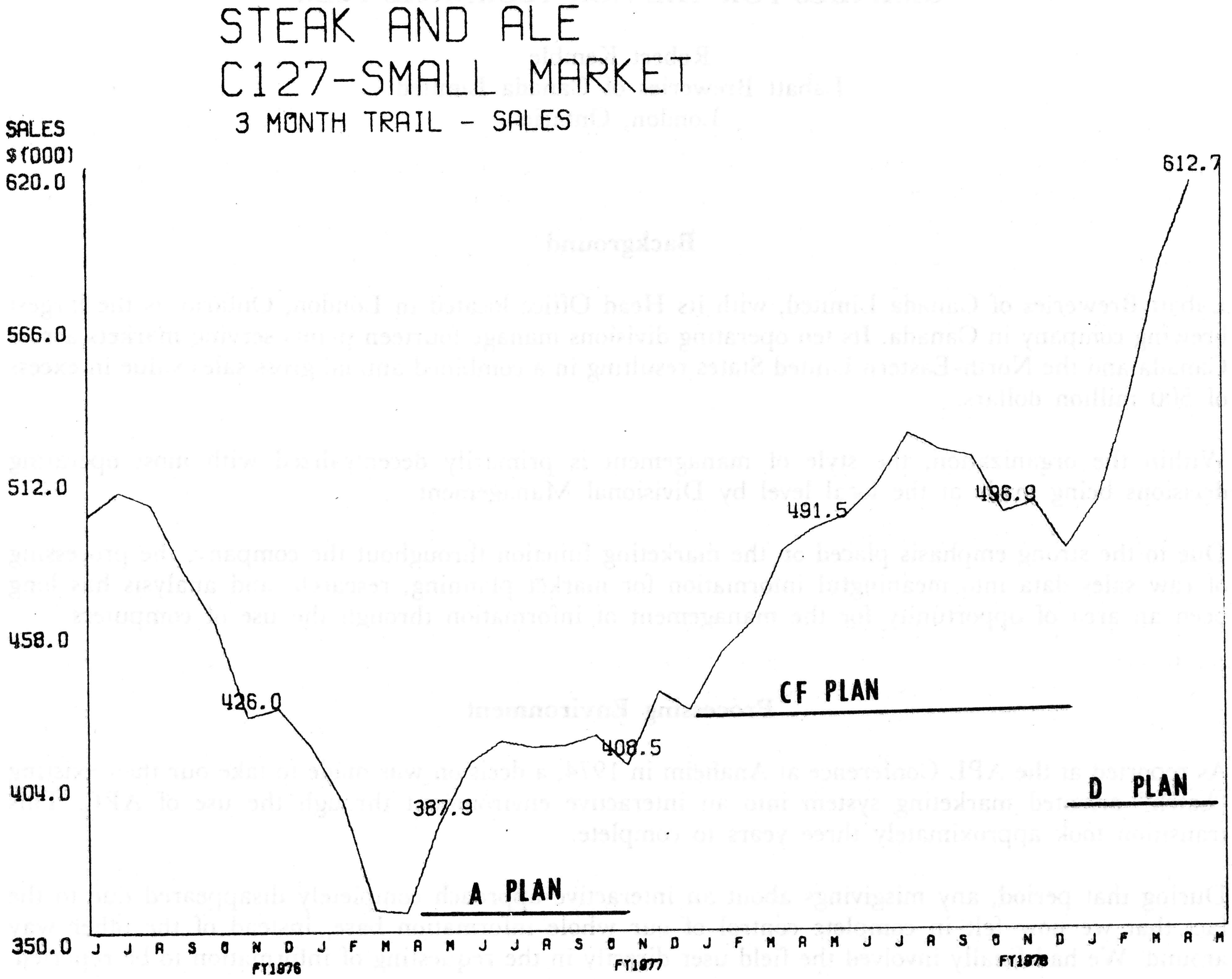
How We Use Graphics

As mentioned previously, we had two main concerns when we originally decided to develop graphics. The first concern was to monitor performance of individual restaurants and the second was to plan and analyze marketing strategies. APL graphics has met our needs in these and other areas.

With the flexibility of our graphics system, executives can request information in many forms. If they are interested in the results of one restaurant, they can request data on that restaurant. Perhaps they are interested in the restaurant's performance after a change in the manager or regional manager. Maybe competition has increased or we have opened a new Steak and Ale in the city. They may want to see the restaurant's performance after a menu change or a price increase. Sometimes they may want to see graphs of all units where sales have fallen from some earlier level. New concepts and new building designs may be closely monitored. The list could go on and on, but the major point is that our executives can see virtually any type of graph for whatever reasons they choose by a simple request.

Our Marketing Department is probably the largest user of the graphics system. Figure 4 shows a typical way that they might use the graphs. After plotting sales for a market, our marketing people superimpose the timing of advertising campaigns on the graphs. This gives them a way to judge the effectiveness of the campaign. Based on these results, they can then plan future advertising and promotional campaigns. These graphs also help to explain the results and the significance of advertising to our operational people.

FIGURE 4



In the areas of financial analysis and planning we are using results from the graphics/trail system to determine budgets, develop seasonal patterns and make forecasts. We analyze the results of restaurant expansions and watch for clues to indicate developing problems. We also use the generalized graphic capabilities to plot charts for in-house manuals and presentations. This is much easier than graphing data manually or sending it out of house for professional-looking results.

Conclusion

No matter how good a system looks on paper, its real test is how well it performs in practice. The graphics/trail system designed for us by I.P. Sharp has become very popular at Steak and Ale. It is designed to function smoothly even for users who have no understanding of programming. The programs which run the system are all conversational. A manual has been prepared which instructs the user how to access Sharp, how to operate the plotter, etc. New data bases can be added with a minimum of effort by Steak and Ale. The monthly update of all data bases only takes about 30 minutes. We are well pleased with the flexibility and efficiency of APL graphics.

GRAPHICS FOR THE NON-TECHNICAL USER

Robert Kemble
Labatt Breweries of Canada Limited
London, Ontario

Background

Labatt Breweries of Canada Limited, with its Head Office located in London, Ontario, is the largest brewing company in Canada. Its ten operating divisions manage fourteen plants serving markets across Canada and the North-Eastern United States resulting in a combined annual gross sales value in excess of 500 million dollars.

Within the organization, the style of management is primarily decentralized with most operating decisions being made at the local level by Divisional Management.

Due to the strong emphasis placed on the marketing function throughout the company, the processing of raw sales data into meaningful information for market planning, research, and analysis has long been an area of opportunity for the management of information through the use of computers.

Processing Environment

As reported at the APL Conference at Anaheim in 1974, a decision was made to take our then existing "batch" oriented marketing system into an interactive environment through the use of APL. This transition took approximately three years to complete.

During that period, any misgivings about an interactive approach completely disappeared due to the fact that we now felt in complete control of our whole information base, instead of the other way around. We had finally involved the field user directly in the requesting of information to be reported.

History of Graphing

We have always believed that information presented graphically would portray marketing data more concisely and more dramatically than conventional numeric means. To that end, a significant amount of time has been spent designing and completing a variety of graph formats manually over the past few years. After the implementation of the interactive marketing system, we felt that many of these manually prepared graphs could be eliminated through the use of computer graphing techniques.

Phase 1

Our initial approach was to add a discrete point graphing capability (scattergram) to the information retrieval workspaces. This addition simply involved the trapping of the data produced for numeric reporting and then passing it on to a graphing function. Our field users' terminal equipment at this time was of a non-graphing style so we were forced to use a scattergram approach. The user, in addition to answering the normal information retrieval prompts, was required to also answer certain key graphing prompts:

1. Titling.
2. Plot character assignments.
3. Plot range, etc.

The addition of the graphing tool to our interactive system was very successful in exposing our field users to the potential of computer graphics. This particular approach, however, did have a number of shortfalls:

- Scattergrams were not considered to be adequate for presentation purposes.
- The user-supplied prompts left too many variables to chance, and in many cases, produced inadequate graphs.
- The influence of graphing highlighted restrictions in our information-retrieval software.
- The cost of the graphs were considered too high.

Phase 2

The limitations of using a non-graphing terminal approach led us, in 1976, to initiate a series of experiments with a Tektronix 4015 terminal and an attached copier in a Service Bureau environment.

Initially we produced time-series graphs of our performance statistics, aggregated and selected by a variety of geographic and product classification criteria. The environment established in this process was one of the field user specifying the graph requirements and a data processing professional extracting the data and preparing the graph in a variety of continuous point plot formats.

This form of graphing generated a high level of interest in that one could readily "see" the developing trends contained within the data.

Concurrent to this work, we also started developing graphing tools for the market research oriented functions. This included the incorporating of demographic and life-style information directly into our interactive report process. This information came from a number of private industry and on-line public data bases like Cansim, General Census Data Base, and the International Monetary Fund (IFS).

For example, we prepared analyses of non-sales information aggregated over census tracts correlated with our sales data. A few tries at producing these reports in a meaningful manner in numeric form were unsuccessful, as the overall impact of the information was absent. It is in this area that our venture into graphic terminals began to pay off.

We prepared on the computer, regional maps by province and major market areas based on available Statistics Canada geostatistical defined maps. By the intelligent use of "hatching", we placed results obtained by information retrieval functions into them, giving us at once an impact study of performance for the area in question.

Once again this process required our users to direct a data processing professional to achieve the required results.

The success achieved with this style of computer graphics had to be tempered with the following disadvantages:

- Reduced involvement of our field users and the corresponding increased work for the data processing professionals.
- Lack of information security in a Service Bureau environment.
- Cost per graph was considered high.
- Specialized information retrieval software required to be created for many of the user requests.

Phase 3

In 1977, a complete changeover to graphing-style terminals was accomplished in our Divisional Offices. These hardware changes coupled with a major overhaul of our information retrieval software allowed us to implement continuous point plot graphing capabilities into our field users' hands. The information retrieval revisions incorporate the use of "MAGIC" which subsequently allows us to use "SUPERPLOT" as well as other I.P. Sharp graphing packages.

Once again our approach was to incorporate additional prompts to the information retrieval software to accommodate the graphing routines. However, this time, almost all of the graphing variables are handled by the software, with the user only providing directional control on such things as:

1. Plot style - line graph, histogram, scattergram, etc.
2. Time series points - monthly, quarterly, etc.
3. Optional plot lines - moving average line, mean value line, etc.

The results of this advance in our use of computer graphics has been very positive. We are enjoying a high level of user acceptance and have for the most part eliminated all of the previous shortcomings in our past approaches. The only exceptions to this statement were those functions accommodated on the Tektronix 4015 for the market research functions in the graphic mapping and "Venn" diagram areas. These were not handled until phase 4.

Phase 4

In 1978 we acquired a Hewlett Packard (HP) 7221-A four-colour flat bed plotter. We are currently adjusting our information retrieval functions in the same manner as discussed in the previous section to accommodate the required graphic prompts to allow the interface with the HP unit. These changes will accommodate the normal time-series graphing function with the added advantages of a pen plotter. We feel we have now finally achieved report presentation level material.

In addition to these developments we are currently working with I.P. Sharp personnel to expand our library of divisional market maps as well as a variety of "Venn" diagram formats for use in the presentation of demographic and life-style data.

Conclusion

The pattern of advancement we have established is to try new ideas or equipment in a controlled, centralized environment with field user direction. Upon satisfactory completion of these experimental developments, we then transfer the capability completely into the field users' hands and monitor the field users' acceptance of the new tools identifying any shortcomings which in turn become the basis for the next development cycle.

This approach in the computer graphics area has brought us now to the level where our non-technical marketing staff feel equally comfortable extracting a numeric marketing report or a computer graph from our interactive marketing systems.

AN INTERACTIVE HARDCOPY GRAPHICS PACKAGE

Alan Harrison
Joe Friswell
British Nuclear Fuels Ltd.
Seascale, Cumbria, England

Introduction

The purpose of this paper is to describe the various facilities available within the Windscale Interactive Graphics System. The package has been written to exploit the facilities of an Anderson Jacobson AJ-832 hardcopy graphics terminal. The system, written in APL, is available on both our major bureaux - I.P. Sharp Associates and Atkins Computing Services Limited.

Why Develop a Windscale Package?

The major problem with interactive graphics has been the high capital cost of obtaining the necessary equipment. In the past few years various hardcopy terminals, which are only slightly more expensive than the teletype style terminal, have become available. However, the major drawback of these has been the expense of using them with standard graphics packages written for the cathode ray tube graphics terminal. The CRT terminal is an intelligent device in that it will draw lines between a pair of coordinates supplied by the computer. To draw an equivalent line the hardcopy terminal requires the generation of all points between the coordinates. This involves a great deal more output which makes the terminal more expensive and slower to run. To overcome the high running cost, the Operational Research Section of the Management Services Group at Windscale developed a package specifically to generate graphs quickly and economically on an AJ-832.

System Details

Basically, the system does not attempt to draw lines but plots points in the usual non automated manner. The points may then, if necessary, be joined by the operator.

Similarly, the full axes are not drawn by the machine. The computer plots the positions of the axis calibration points and their numeric value, up to a maximum of six characters per point. (If any information is lost by this curtailment, the complete values are printed below the graph.)

The increment between calibration points will be related to the range of the data points, being first the power of 10 less than the range and then successively halved if insufficient points would appear, so that ultimately between 4 and 10 calibration points will be marked on each axis.

Examples: (a) Points : 1 3 10
Calibration : 1 2 3 4 5 6 7 8 9 10

(b) Points : 1 3
Calibration : 1 1.5 2 2.5 3

Syntax and optional commands are discussed in section 4 and may be used either interactively or within a user-written program. A number of global variables, the names of which all begin GV...., record the state of the system and should not be overwritten by the user.

Examples of the graphs are contained in Appendix I: Examples.

User Details

1. Loading the Graphics Package

The graphics system is kept as a workspace on both the I.P. Sharp and Atkins On-line bureaux.

It is also supported as a package variable (on the IPSA system only) to enable it to be loaded under program control. Details of access may be obtained from the authors.

2. Plotting Functions (examples of the use of these functions are in Appendix 1)

2.1 *PLOT*

Syntax: *R←X PLOT Y*

The basic plotting function, *PLOT*, prints a single graph, the plotting character for which is the first element of the vector of available plot characters. Like the other plotting functions, it returns an explicit result (see section 3).

The left-hand and right-hand arguments are vectors of the x-coordinates and y-coordinates respectively.

2.2 *MULTIPLOT*

Syntax: *R←X MULTIPLOT Y*

The intermediate plotting function *MULTIPLOT* will plot several graphs on the same axes, but each must have the same number of points.

As with *PLOT*, the left and right parameters are the x- and y-coordinates respectively, but their shapes may vary:

a) If both x and y are vectors, a single graph is drawn (equivalent to using *PLOT*).

b) If either x or y is a vector and the other a matrix, each row of the matrix is plotted, with a different character, against the common values of the other coordinates given in the vector. The length of the vector must equal the number of columns of the matrix, and is the number of points plotted on each graph.

c) If both x and y are matrices then they must be of the same shape and each row of x is plotted against the corresponding row of y.

2.3 *SUPERPLOT*

Syntax: *SUPERPLOT NAMES*

The most versatile of the plotting functions, *SUPERPLOT*, is monadic and will plot one or several graphs of the same or differing lengths on the same axis. Any graph that can be plotted by either *PLOT* or *MULTIPLOT* can also be plotted by *SUPERPLOT*.

The parameter *NAME*, is a character vector of a series of variable names, delimited by blanks, commas or both. All the variables must be vectors and the first one is taken as x-coordinates to be plotted against the second taken as y-coordinates. Similarly, the third named vector is plotted against the fourth, and so on. (In general, any APL expression not containing a delimiter may be used as an element of *NAME*.)

3. The Explicit Result

The plot, if it is generated, is returned as an implicit result. An explicit result is also returned, which may be either

(a) An empty vector - meaning that the plot has been generated correctly.
(b) An error message - signifying that the plot could not be generated as specified, and has either been aborted, or been printed lacking some information.

The messages currently supported are shown in Appendix 2.

By testing the shape of the explicit result of the plotting function, a user program may deduce whether the graph has been correctly produced, e.g., `→ERROR×⍳(0≠⍴X PLOT Y)` will transfer control to label *ERROR* if an error message causes the returned result to be other than an empty vector.

4. Options

4.1 *DEFAULT*: resets the package to the status described in section 5 using the default options noted below.

4.2 *TYPE A*: determines the type of axes the graph is to have, and whether or not the data is to be used to construct a histogram. Parameter *A* may be:

(1) *ZEROAXES* - the graphs generated have axes with their origin at (0; 0) (or at 1 on a log-scale axis).

(2) *BASEAXES* - the graphs have axes, the origin of which is formed from the minimum value of each coordinate set (see also option *RANGE*, section 4.6).

(3) *NOAXES* - as *BASEAXES*, except that no axes at all are printed.

(4) *HISTOGRAM* - graphs are generated as histograms or bar-charts, with printed axes and an origin (0; 0). Coordinate pairs are taken as the top right-hand point of each bar, and the top left-hand point is generated at the previous y-coordinate. Care should be taken over the interpretation of the axis calibration when using this option.

Default: *TYPE ZEROAXES*

4.3 *SWIVEL*: will rotate the printed graph from the default orientation (see Figure 1(a)) through a clockwise right-angle to the orientation of Figure 1(b). Repetition of swivel will restore the default orientation.

Figure 1(a)
Normal Orientation

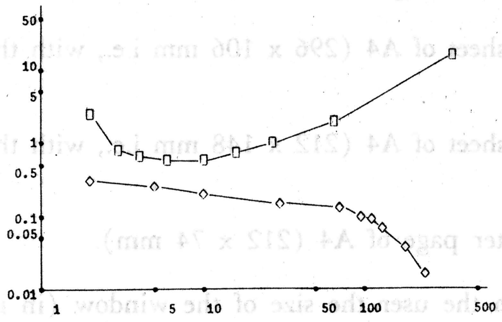
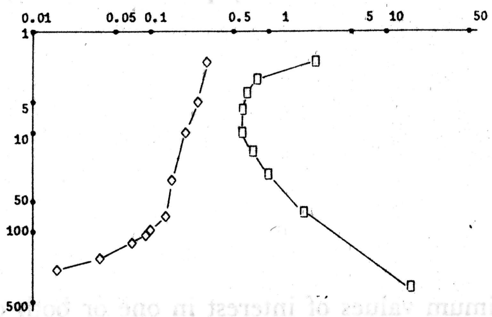


Figure 1(b)
Swivelled Orientation



Default: As Figure 1(a)

4.4 *SIZE A*: adjusts the size of the window in which the graph will be plotted. Parameter *A* may be:

(1) *FULLGRAF* - the graph is scaled to fit a standard 340 mm wide by 215 mm deep line-printer page.

(2) *A4GRAF* - scaled to fit standard A4 paper (296 mm wide by 212 mm deep) for easier copying and filing.

(3) *HALFGRAF* - scaled to fit a half sheet of A4 (296 x 106 mm i.e., with the long axis horizontal).

(4) *SQUARGRAF* - scaled to fit a half sheet of A4 (212 x 148 mm i.e., with the long axis vertical).

(5) *QUARTGRAF* - scaled to fit a quarter page of A4 (212 x 74 mm).

(6) *SPECIALGRAF* - which requests from the user the size of the window (in inches) and scales the graph to fit it.

Default: *SIZE A4GRAF*

4.5 *AXIS TITLE A*: prints the character vector *A* as

(1) the graph title centred over the graph, if *AXIS* is absent

(2) an axis title, positioned as to annotate the axis described by parameter *AXIS*, which may be

(i) *XAXIS*
(ii) *YAXIS*

Default: *TITLE ''*
XAXIS TITLE ''
YAXIS TITLE ''

4.6 *AXIS RANGE A*: sets the minimum and maximum values of interest in one or both coordinates. Option *TYPE BASEAXES* (q.v.) is selected automatically.

Parameter *AXIS* may be either

(1) *XAXIS* - limits are to apply to the x-coordinates.
(2) *YAXIS* - limits are to apply to the y-coordinates.

Parameter *A* may be

(1) *AUTO* - the default value which causes the range of plotting to be computed from the data.
(2) A vector of 2 numbers - which specify, respectively, the minimum and maximum values of the coordinate range.

The *RANGE* option can be used to exclude data outside the range of interest because of error, to plot different sets of data on directly comparable axes, and to bias the axis as is done using *TYPE BASEAXES*.

Default: *XAXIS RANGE AUTO*
YAXIS RANGE AUTO

4.7 *AXIS SCALE A*: is used to plot the data on a logarithmic scale in one or both dimensions.

Parameter *AXIS* specifies which dimension is referred to, and may be

(1) *XAXIS*
(2) *YAXIS*

Parameter *A* may be:

(1) *LOG* - which specifies a logarithmic scale.
(2) *LIN* - the default, which returns the scale to the normal linear pattern.

Example: *YAXIS SCALE LOG* will cause the basic increment in the vertical axis to represent a ten-fold multiplication in coordinate value.

Default: *XAXIS SCALE LIN*
YAXIS SCALE LIN

4.8 *PLOTCHAR A*: where *A* is a character vector of the character(s) which will be used in plotting subsequent graphs. Graphs generated using *PLOT* will always use the first character of A, and a set of graphs generated by *MULTIPLOT* or *SUPERPLOT* will use the appropriate number of characters from the front of *A*.

The positioning of characters such as full-stop, "." is adjusted before printing to give graphical accuracy.

Default: *PLOTCHAR* '+.×∘⋆◇□∇∆*X*'

4.9 *PROMPT A*: affects the printing of the prompting message

ALIGN PAPER AND HIT RETURN-KEY :

before generating each graph. Valid values of *A* are:

(1) *QUIET* - in which case the printing is suppressed, the print-head indents 12 spaces and the terminal awaits a carriage return.

(2) *LOUD* - the default value, which restores the printing of the prompting message.

The use of the *ALIGN* option to print the graph other than at the top of the page selects *PROMPT QUIET* automatically.

Default: *PROMPT LOUD*

4.10 *ALIGN A*: controls whereabouts on the page the graph is printed.

Parameter *A* may be:

(1) *MANUAL* - the printer stops and awaits manual adjustment of the paper and a carriage return signal before printing the graph.

(2) *UNDER* - the printer is aligned below the graph printed on the previous page, and awaits a carriage return.

(3) a scalar N - the printer will be aligned at the left-hand margin of the previous page and N inches down it.

When either of the special options (2) or (3) is selected,

(a) option *PROMPT QUIET* is enforced automatically, and
(b) if the size of the graph to be plotted exceeds the remaining space on the page, the special option is overruled and the default recalled.

Also, the *ALIGN* option resets to the default status after every graph, unlike other options which require a positive statement to change status.

Default: *ALIGN MANUAL*

4.11 *DEFAULTPAGE*: sets the page length to 51 lines (the size of the standard paper used), the page width to 140 characters, and sets the top-of-page marker (form feed origin) to the present line.

This option is chosen automatically on loading the plotting workspace.

4.12 *PAGELENGTH A*: sets the vertical form length to *A* lines, where *A* may be the range `0 < A < 127`, and marks the present line as the form feed origin.

Default: *PAGELENGTH* `51`

4.13 *PAGEWIDTH A*: sets the horizontal form width to *A* characters, where *A* may be in the range `0 < A < 155`, with the left-hand margin at position `0` and the right-hand at position *A*.

Default: *PAGEWIDTH* `140`

5. Default Settings

The default settings as noted under each option in section 4 are operative when the system workspace is loaded. This combination may also be returned after using other options by exercising the *DEFAULT* option.

In summary, when one of the plotting functions is called, the default settings will cause the alignment prompt to be produced, following which the printer will await a carriage return to signify that the user has aligned the paper to his satisfaction. On printing, the graph will be of a size to fit on A4 paper with the x-axis horizontal (across the paper). No titles are printed and the plotting character for the first graph is "+".

The axes will have an origin at `(0; 0)` and be calibrated in linearly spaced increments up to a maximum sufficient to include the highest value of each coordinate.

Although the graph is scaled to A4 size, the page size is set to standard line printer form size (51 lines x 140 characters), and a form feed origin is generated at the top left corner of each graph printed.

Acknowledgements

The views expressed are the personal views of the authors and we would like to acknowledge our thanks to BNFL for providing the opportunity to give this paper.

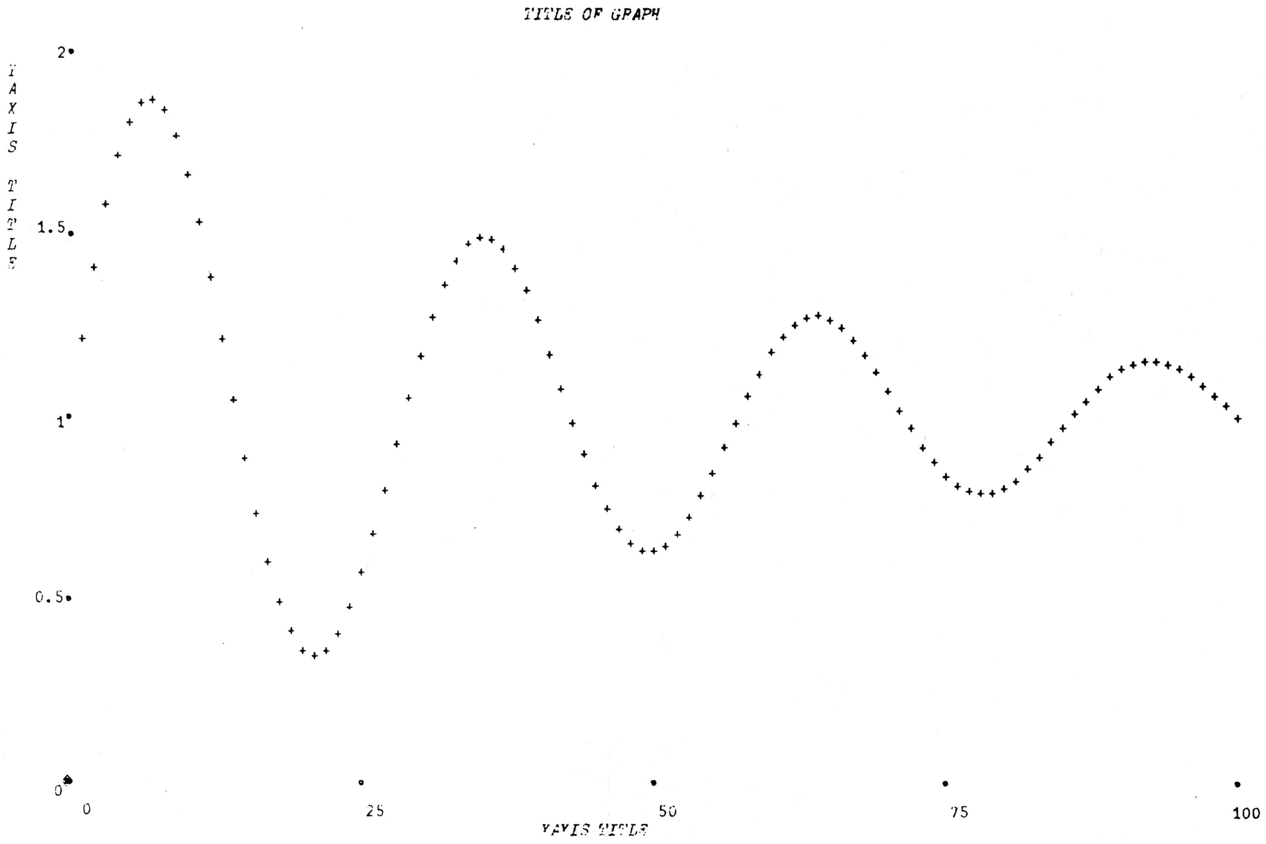
Enquiries on the content and possible use of this package should be directed to Alan Harrison, Management Services Group, Windscale and Calder Works, Seascale, Cumbria, England.

Appendix 1: Examples

All the examples within this appendix have been generated under program control, but they could equally well have been produced interactively. Example 1 could have been generated by typing lines `[7]`, `[8]`, `[9]`, and `[10]` while in "desk-calculator" mode.

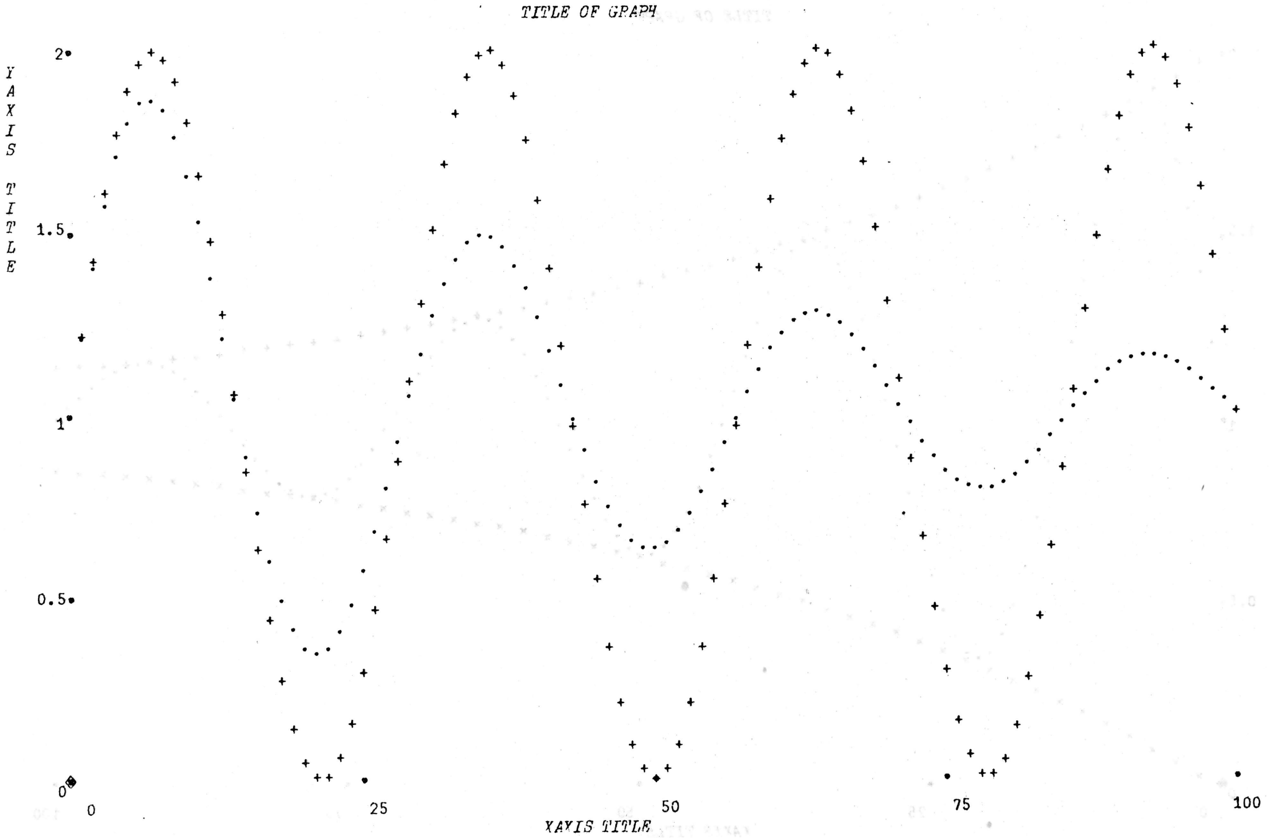
Example 1

```
     ∇ EXAMPLE1
[1]   ⍝ THIS IS A SIMPLE PLOT OF A DAMPED SINE-WAVE TO ILLUSTRATE :
[2]   ⍝    (1)  FUNCTION 'PLOT',
[3]   ⍝    (2)  THE CALIBRATION OF THE AXES, AND
[4]   ⍝    (3)  THE POSITION OF THE TITLES
[5]   ⍝ THE OTHER TWO PLOTTING FUNCTIONS ARE DEMONSTRATED IN THE NEXT TWO EXAMPLES
[6]   ⍝
[7]    TITLE 'TITLE OF GRAPH'
[8]    XAXIS TITLE 'XAXIS TITLE'
[9]    YAXIS TITLE 'YAXIS TITLE'
[10]   DAMPSINEX PLOT DAMPSINEY
     ∇


ALIGN PAPER AND HIT RETURN-KEY :
```



Example 2

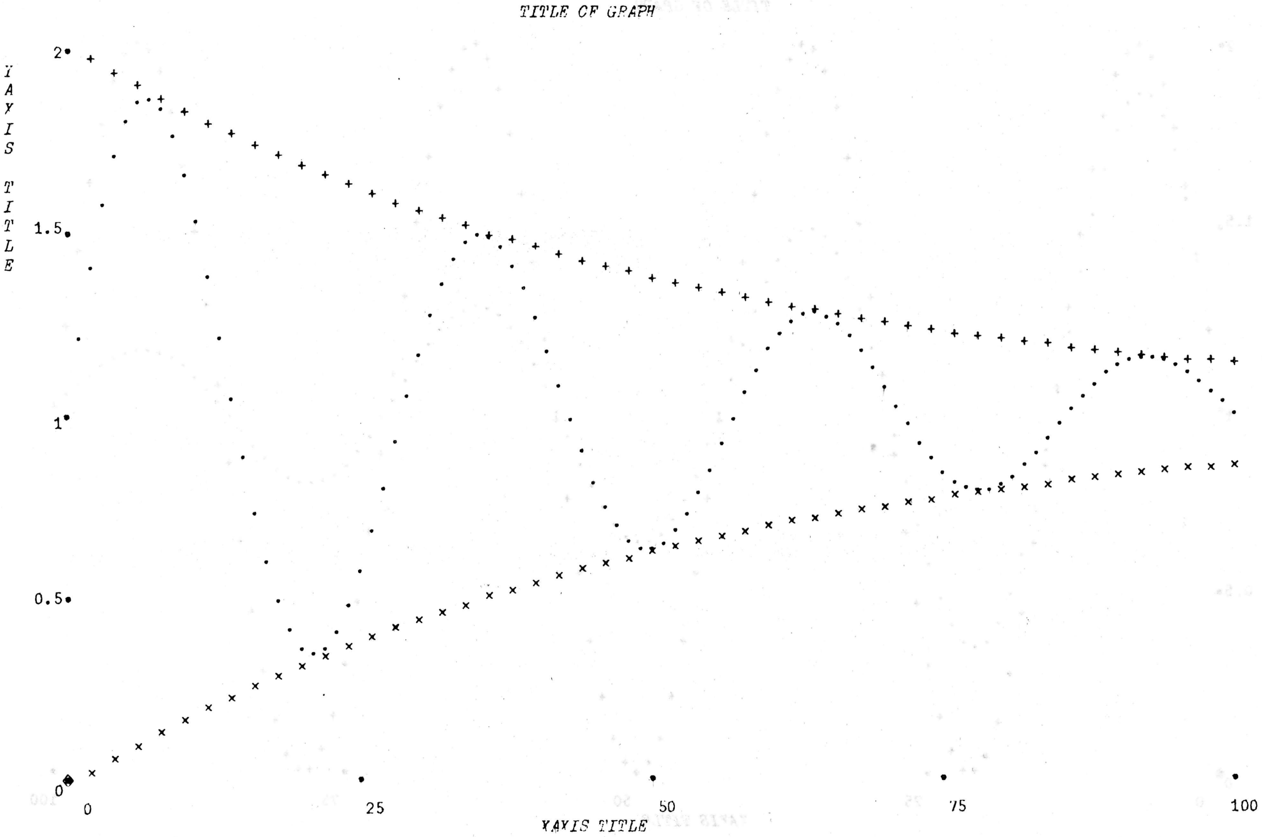
```
      ∇ EXAMPLE2
[1]    ⍝ TWO GRAPHS OF THE SAME LENGTH, AND, IN THIS CASE,
[2]    ⍝ WITH THE SAME X-COORDINATES, CAN BE PLOTTED TOGETHER
[3]    ⍝ USING 'MULTIPLOT'
[4]    ⍝
[5]      DAMPSINEX MULTIPLOT(2,⍴SINEY)⍴SINEY,DAMPSINEY
      ∇

ALIGN PAPER AND HIT RETURN-KEY :
```



Example 3

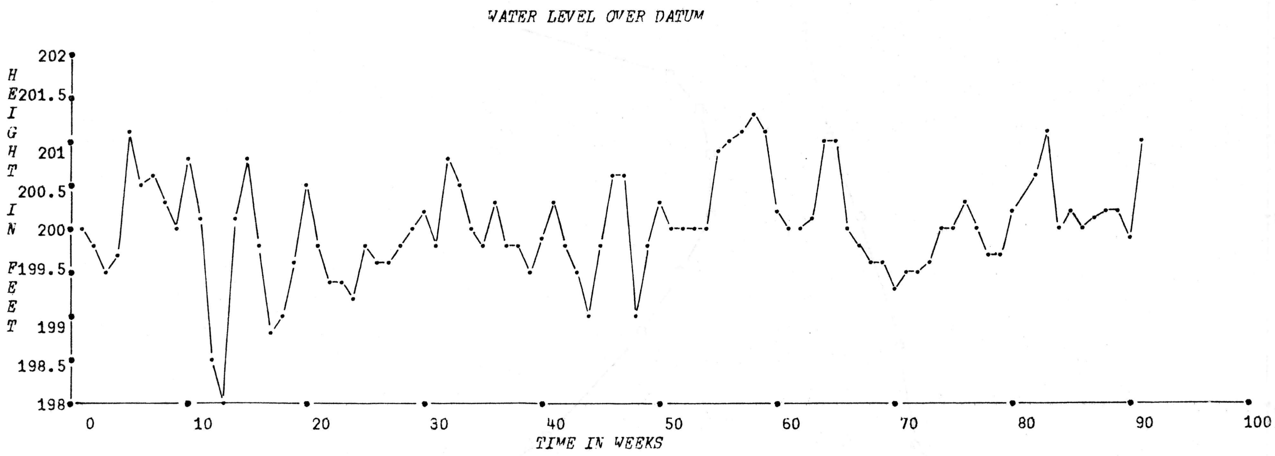
```
    ∇ EXAMPLE3
[1]   ⍝ WHEN THE LENGTHS OF THE GRAPHS DIFFER 'SUPERPLOT' MUST BE USED
[2]   ⍝
[3]     SUPERPLOT 'ENVX,ENVHI,DAMPSINEX,DAMPSINEY,ENVX,ENVLO'
[4]     DEFAULT
    ∇
```

```
ALIGN PAPER AND HIT RETURN-KEY :
```



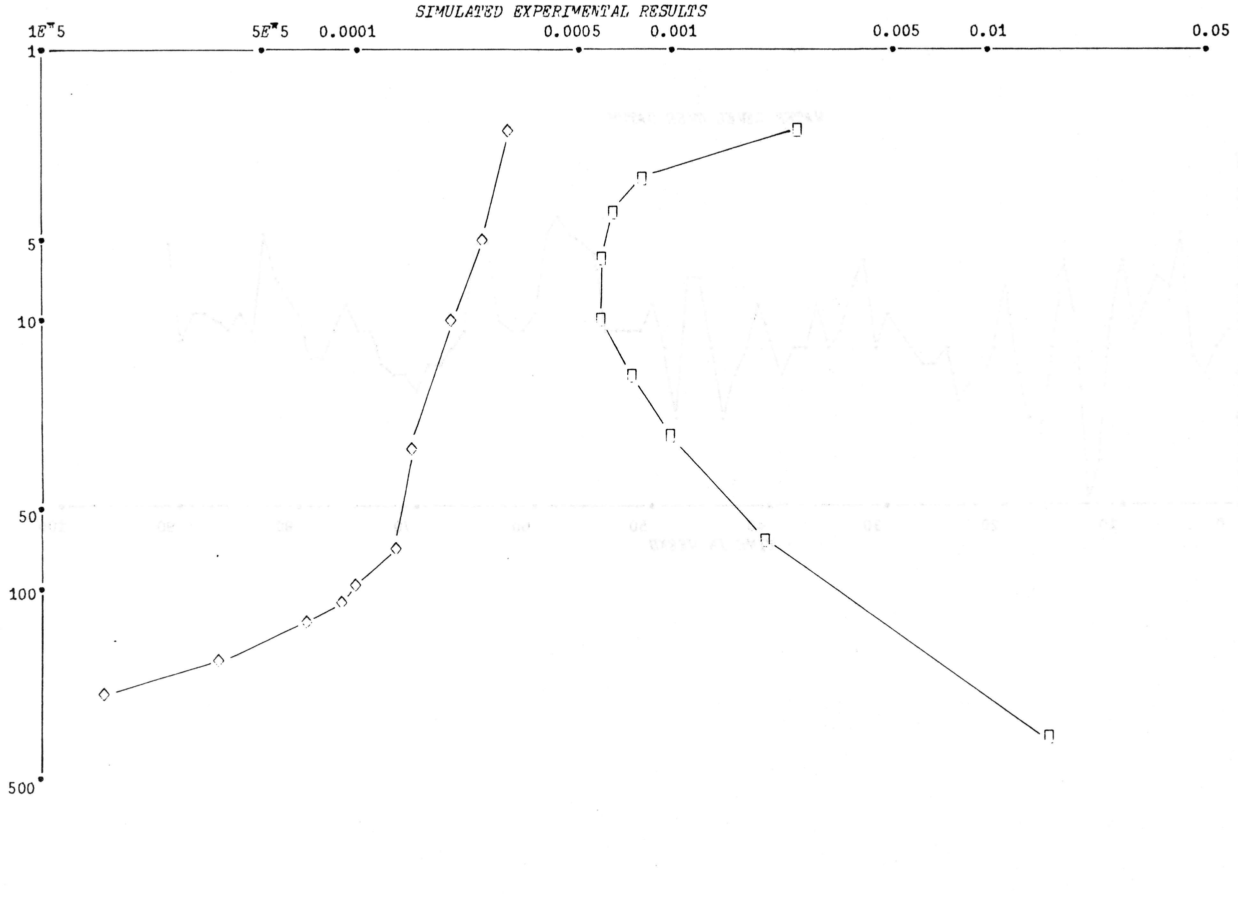
Example 4

```
     ∇ EXAMPLE4
[1]   ⍝ SOME GRAPHS, FOR EXAMPLE LONG TIME-SERIES, ARE MOST EFFECTIVELY
[2]   ⍝ IN A LONG SQUAT FORMAT.
[3]   ⍝ (THE JOINING OF POINTS AS SHOWN ON THIS AND SUBSEQUENT GRAPHS MUST BE DONE BY HAND.)
[4]   ⍝
[5]    TITLE 'WATER LEVEL OVER DATUM'
[6]    XAXIS TITLE 'TIME IN WEEKS'
[7]    YAXIS TITLE 'HEIGHT IN FEET'
[8]    YAXIS RANGE 198 202
[9]    SIZE HALFGRAF
[10]   PLOTCAR '.'
[11]   WATERTIME PLOT WATERLEVEL
[12]   DEFAULT
     ∇

ALIGN PAPER AND HIT RETURN-KEY :


DISPLAYED
```



Example 5

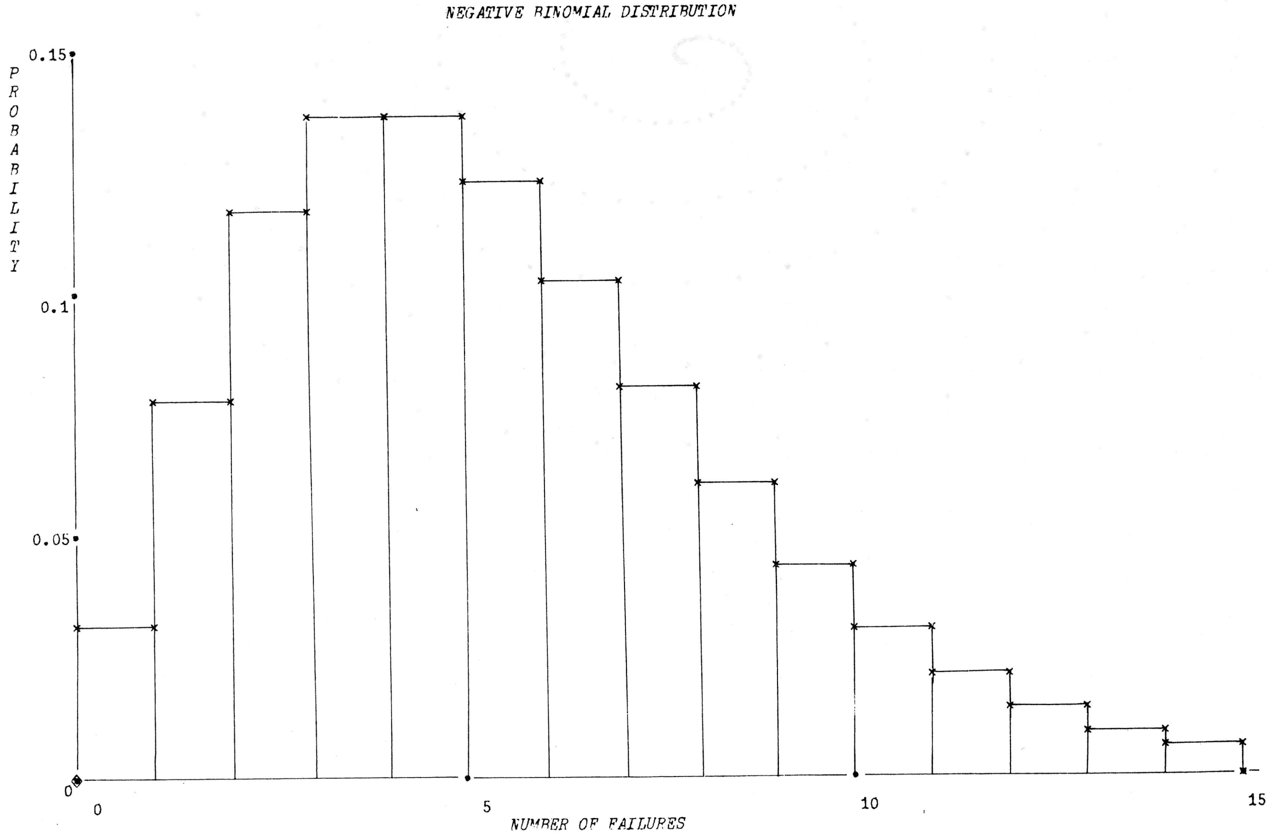
```
    ∇ EXAMPLE5
[1]   ⍝ OTHERS, HOWEVER, ARE MORE SUITED TO A TALL FORMAT,
[2]   ⍝ BUT PAPER PERFORATIONS MAY BE OBTRUSIVE IF THE LENGTH OF GRAPH
[3]   ⍝ IS GREATER THAN THE PAPER DEPTH.
[4]   ⍝ TO ALLEVIATE THIS, THE GRAPH ITSELF (BUT NOT THE TITLING ETC.)
[5]   ⍝ MAY BE SWIVELLED THROUGH A RIGHT-ANGLE.
[6]   ⍝
[7]    TITLE 'SIMULATED EXPERIMENTAL RESULTS'
[8]    XAXIS SCALE LOG
[9]    YAXIS SCALE LOG
[10]   PLOTCHAR '⎕◊'
[11]   TYPE BASEAXES
[12]   SWIVEL
[13]   SUPERPLOT 'EXP1X,EXP1Y,EXP2X,EXP2Y'
[14]   DEFAULT
    ∇

ALIGN PAPER AND HIT RETURN-KEY :
```



Example 6

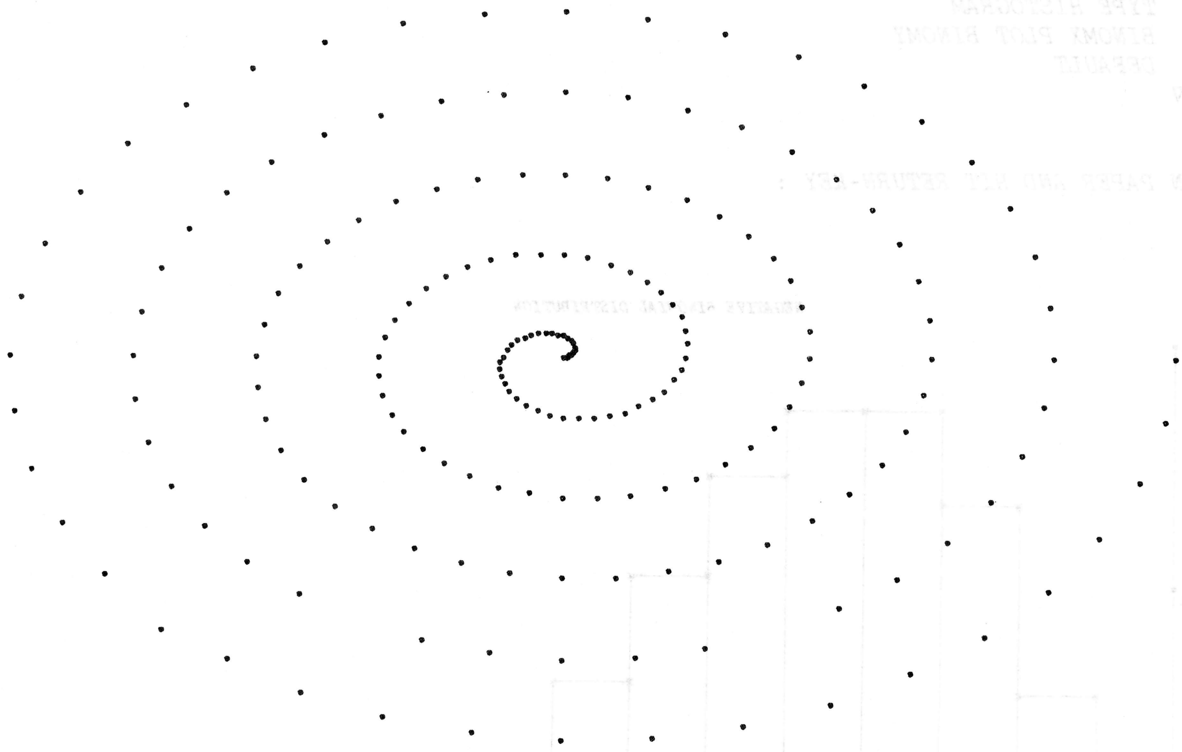
```
    ∇ EXAMPLE6
[1]   ⍝ HISTOGRAMS OR GROUPED DATA CAN BE DISPLAYED IN BLOCK FORM,
[2]   ⍝ USING THE HISTOGRAM OPTION.
[3]   ⍝ (IT IS ASSUMED THAT THE FIRST CLASS OF DATA EXTENDS FROM ZERO
[4]   ⍝ TO THE FIRST ABSCISSA COORDINATE, THE SECOND CLASS BETWEEN
[5]   ⍝ THE FIRST AND SECOND COORDINATES, ETC.,
[6]   ⍝ AND THE HEIGHT OF THE BLOCK IS MARKED AT EACH END OF IT.
[7]   ⍝ CARE MUST BE TAKEN OVER THE INTERPRETATION OF THE CALIBRATION NUMBERS,
[8]   ⍝ SINCE THEY CAN REFER TO EITHER OF THE BLOCKS TO RIGHT OR LEFT OR THE
[9]   ⍝ CALIBRATION POINT. IN THIS CASE THEY REFER TO THE RIGHT,
[10]  ⍝ I.E.  Y(0)  IS THE FIRST BLOCK.)
[11]  ⍝
[12]   TITLE 'NEGATIVE BINOMIAL DISTRIBUTION'
[13]   XAXIS TITLE 'NUMBER OF FAILURES'
[14]   YAXIS TITLE 'PROBABILITY'
[15]   PLOTCHAR '×'
[16]   TYPE HISTOGRAM
[17]   BINOMX PLOT BINOMY
[18]   DEFAULT
    ∇
```

```
ALIGN PAPER AND HIT RETURN-KEY :
```



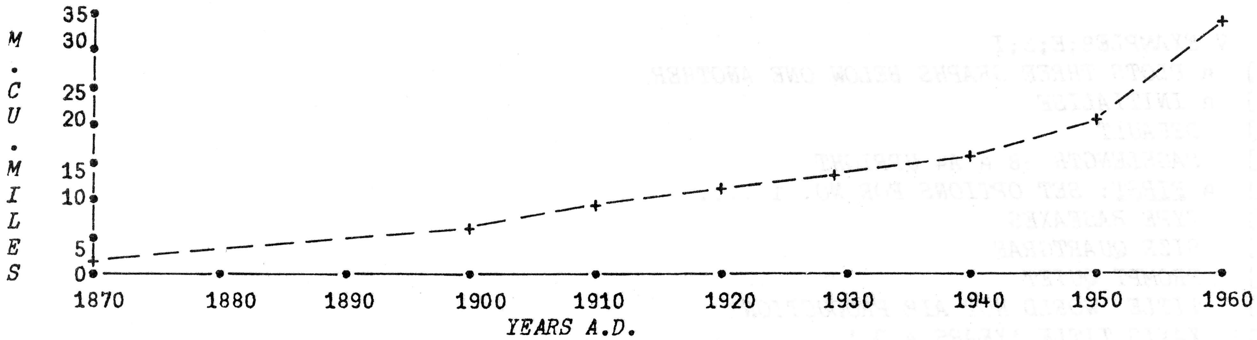
Example 7

```
     ∇ EXAMPLE7
[1]   ⍝ DECORATIVE PLOTS DO NOT NECESSARILY NEED A REFERENCE FRAME.
[2]   ⍝
[3]     PLOTCHAR '∘'
[4]     TYPE NOAXES
[5]     SPIRAL[1;] PLOT SPIRAL[2;]
[6]     DEFAULT
     ∇

ALIGN PAPER AND HIT RETURN-KEY :
```

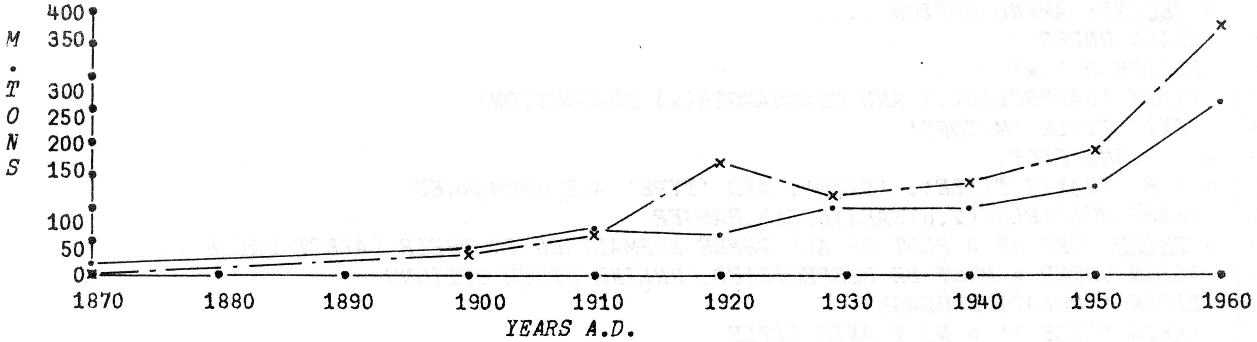


Example 8

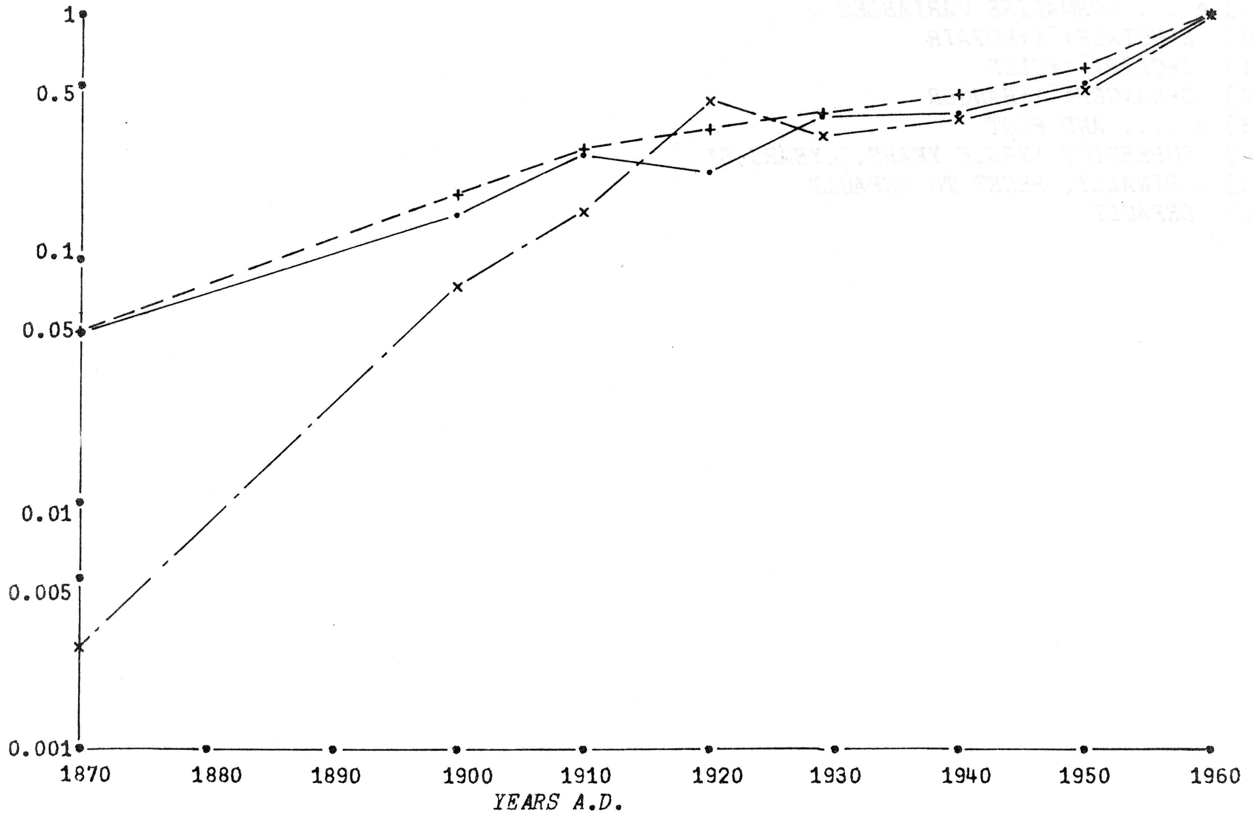
```
    ∇ EXAMPLE8;E;S;I
[1]   ⍝ PLOTS THREE GRAPHS BELOW ONE ANOTHER
[2]   ⍝ INITIALISE
[3]    DEFAULT
[4]    PAGELENGTH 68 ⍝ A4 UPRIGHT
[5]   ⍝ FIRST: SET OPTIONS FOR NO. 1 ....
[6]    TYPE BASEAXES
[7]    SIZE QUARTGRAF
[8]    PROMPT QUIET
[9]    TITLE 'WORLD HOT AIR PRODUCTION'
[10]   XAXIS TITLE 'YEARS A.D.'
[11]   YAXIS TITLE 'M.CU.MILES'
[12]  ⍝ ....AND PLOT IT
[13]   YRS PLOT HOTAIR
[14]  ⍝ SECOND: AMEND OPTION ....
[15]   ALIGN UNDER
[16]   PLOTCHAR '.×'
[17]   TITLE 'PAPERCLIP(.) AND COATHANGER(×) PRODUCTION'
[18]   YAXIS TITLE 'M.TONS'
[19]  ⍝ ....AND PLOT.
[20]  ⍝ N.B. 'XAXIS TITLE', 'SIZE', AND 'TYPE' ARE UNCHANGED
[21]   YEARS MULTIPLOT(2,⍴YEARS)⍴CLIP,HANGER
[22]  ⍝ THIRD: SET UP A PLOT OF ALL THREE NORMALISED TO THEIR LATEST VALUE ....
[23]   ALIGN UNDER ⍝ MUST BE RESPECIFIED, UNLIKE OTHER OPTIONS
[24]   TITLE ' RELATIVE USAGE'
[25]   YAXIS TITLE '' ⍝ NO Y-AXIS TITLE
[26]   SIZE SQUARGRAF
[27]   YAXIS SCALE LOG
[28]   PLOTCHAR '+.×'
[29]  ⍝ ....NORMALISE VARIABLES ....
[30]   E←HOTAIR÷¯1↑HOTAIR
[31]   I←CLIP÷¯1↑CLIP
[32]   S←HANGER÷¯1↑HANGER
[33]  ⍝ .... AND PLOT
[34]   SUPERPLOT 'YRS,E YEARS,I,YEARS,S'
[35]  ⍝ FINALLY, RESET TO DEFAULT
[36]   DEFAULT
    ∇
```



WORLD HOT AIR PRODUCTION



PAPERCLIP(.) AND COATHANGER(×) PRODUCTION



RELATIVE USAGE

Appendix 2: Error Messages

All plotting functions (section 2 under User Details) return a result which can be either

(a) an empty vector, meaning that the plot has been generated correctly, or
(b) an error message.

A2.1 *ERROR 1 : NUMBER OF X-Y COORDS UNEQUAL. PLOT ABORTED*

Generated within *PLOT* when the right-hand and left-hand parameters are of unequal length. This condition cannot arise when *PLOT* is called by *MULTIPLOT* or *SUPERPLOT* (except when *MULTIPLOT* has two unequal length vector arguments).

A2.2 *ERROR 2 : DATA MATRICES NOT OF EQUAL SHAPE. ABORT*

Generated by *MULTIPLOT* when the right- and left-hand parameters are both matrices, but of differing shapes.

A2.3 *ERROR 3 : UNEQUAL COORDINATE LENGTHS. ABORT*

Generated by *MULTIPLOT* when the length of the vector argument is different from the number of columns of the matrix argument.

A2.4 *ERROR 4 : ODD NUMBER OF NAMES. ABORTED*

Generated by *SUPERPLOT* when the number of plot variables in its parameter is detected to be odd, because there is not an even number of delimiters separating them.

A2.5 *ERROR 5 : MISMATCHED COORDS. OMIT*

Generated by *SUPERPLOT* when the lengths of two vectors selected to be plotted against one another are found to be unequal. This does not cause the plot to be aborted but drops the offending variables from the print. If another error is detected, the total message will consist of a catenation of the individual messages, with embedded line-feeds in a LIFO order.

A2.6 *ERROR 6 : NO POINTS TO PLOT*

Generated by *SUPERPLOT* when all data vectors named in the list are dropped through errors of type A2.5. A plot is not attempted.

Other errors, such as the specification of a matrix as a parameter of *PLOT*, or the exclusion of all data points through misuse of the *RANGE* option will cause a system error and suspend execution, in the usual manner, at the offending line.

Error detection, for example, during production of a standard report, may be acted upon by use of the error code number, for example:

```
→LABEL×⍳('5'=RESULT[7])
```

will transfer control to *LABEL* if, despite the error, a graph has been printed.

THE AUTOMATED INVENTORY SYSTEM

John J. Fuller
Xerox Corporation
Webster, New York

The Automated Inventory System (AIS) is a set of APL programs designed to replace manual inventory procedures at Equipment Logistics Centers (ELC) throughout the United States. In order to appreciate the advantage of an on-line computerized inventory system, one must first understand the logistic problem it is meant to solve.

Xerox Corporation manufactures reprographic equipment in Webster, N.Y. and distributes these machines throughout the United States by way of a network of branch sales offices. Seven Equipment Logistic Centers, where equipment is inventoried, are strategically located throughout the United States. Each ELC receives equipment from manufacturing and reconditioning centers. They store this equipment until it is requested by a branch. The object of having ELC's is to improve customer service by minimizing the delay in machine placement while attempting to gain economies of scale with regard to freight expenses.

ELC's are responsible for millions of dollars worth of inventory, and thus they must maintain careful control of the status of their inventory on a machine basis. Timely information on inventory status enables the ELC to manage its equipment inventory more effectively, reducing inventory cost.

Formerly, this function was accomplished by a series of manual procedures. A card was maintained for each machine in inventory. This card contained information relevant to the machine such as attached accessory equipment, model ID, status, date first received into inventory, date of last status change and sublocation. Sublocation refers to a physically separate storage location (say a particular building) where the machines are actually stored. These card files were kept in serial number order and were referred to when an ELC had questions concerning its inventory. Inquiries as simple as total inventory by product line were ardorous manual tasks of counting and classifying. Often the maintenance of this card file required two clerks full time. Inquiries as simple as finding the serial numbers of all equipment received between two given dates required visual inspection of the entire card file. Clearly this was a function which was ripe for computerization.

However, these were not the only tasks performed manually at the ELC. The company's equipment inventory accounting programs had to be made aware of each machine's location at all times. Thus, the ELC manually prepared keypunch documents (reflecting machines received or shipped) which were sent each day to a regional data processing center for entry into a batch computer system which kept track of equipment.

The batch system issued reports weekly on inventory status which then had to be manually compared to the card file. It was often important for a branch to know which equipment was in transit to it from an ELC. So, in addition to maintaining a card file and creating input documents for the batch inventory program, the ELC would send a Telenet message to the receiving branch listing the serial numbers of machines which were in transit. Basically the ELC was processing the same information on equipment shipped and received three times in separate ways.

The purpose in creating the Automated Inventory System was thus threefold.

1) To provide current inventory status information at any level of detail or aggregation desired.

2) To automatically generate a feed each night to each regional data processing center, consisting of the ELC's ship and receive activity.

3) To automatically create and disseminate messages reflecting daily ELC activity to selected network nodes (generally branch sales offices and region equipment control groups) on an overnight basis.

Automated Inventory System Description

The question I wish to address now is the suitability of APL for satisfying these objectives. What the ELC needed was computer power. Time-sharing seemed a logical candidate for providing such computer power without the problem and overhead associated with operating a computer in the ELC.

The type of operation mentioned earlier, searching the inventory for machines with a given set of characteristics is a problem well suited to APL's array processing power. Xerox operates an IBM 158/3 computer running under the I.P. Sharp Operating System in Webster, N.Y. It also leases various telecommunications facilities including the I.P. Sharp Network, Telenet network and a local hardwired private network called Monroe County Data Network (MCDN). Through these facilities each Equipment Logistic Center (ELC) has access to APL. Each ELC has local network access via dial-up lines. Since the Diablo 1550 terminal is well suited both to data entry and printing reports on standard computer paper, it was placed in each ELC. Due to large activity volume, some ELC's required two terminals. Thus the system had tto handle the problem of properly queuing requests when both tasks attempted to update the data base at the same time.

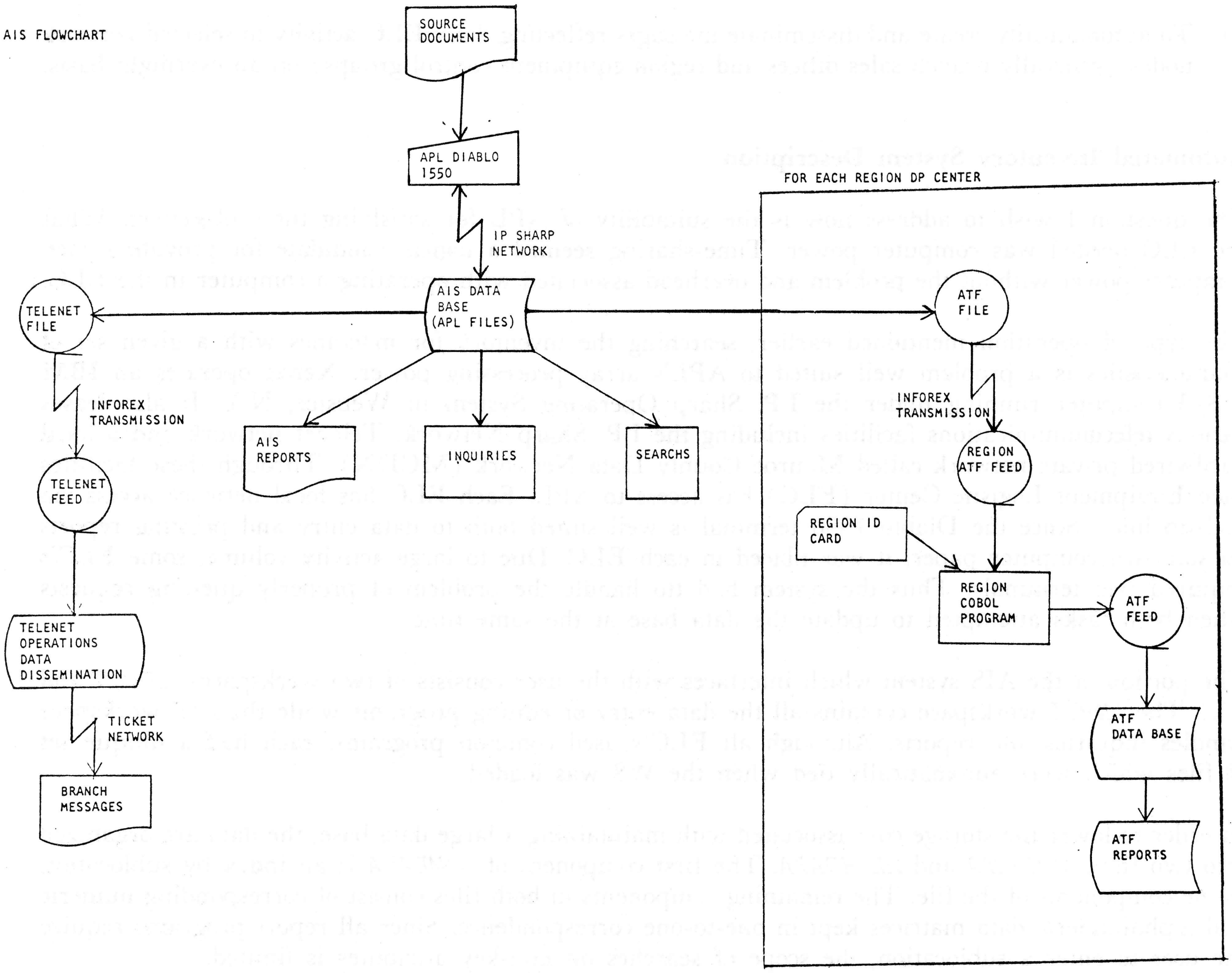
The portion of the AIS system which interfaces with the user consists of two workspaces, *INPUT* and *AIS*. The *INPUT* workspace contains all the data entry or editing programs while the *AIS* workspace handles inquiries and reports. Although all ELC's used common programs, each had a unique set of files which were automatically tied when the WS was loaded.

In order to lower the storage cost associated with maintaining a large data base, the data are organized into two files, *NUMDATA* and *ALPHDATA*. The first component of *NUMDATA* is an index by sublocation of the components of the file. The remaining components in both files consist of corresponding numeric and alphanumeric data matrices kept in one-to-one correspondence. Since all report programs require the user to enter a sublocation, the scope of searches on non-key attributes is limited.

The other two areas where the AIS system reduces manual effort are in supplying a digital feed to the batch accounting system (which was formerly handled by data entry at the regional data processing centers from keypunch documents prepared at the ELC) and in automating the interface (via a tape feed) between the APL system and a private Telenet network. The ELC formerly manually typed a message into the Telenet network on a teletype machine. These messages are notices to branches receiving equipment listing serial numbers of machines which are in transit. The AIS system creates files from data input at the ELC each night. Each night the computer operator initiates a task which creates and dumps to tape a feed for each of five regional data centers. These tapes are Inforexed each night to the proper regional accounting systems. Likewise, a special Telenet feed tape is prepared each night for dissemination over the company's private Telenet network.

Nighttime dissemination of these messages releases network daytime capacity. The reduction of manual steps in the Automated Inventory System (AIS) has resulted in both more timely and more accurate information. Figure 1 gives a flowchart of the information flow within the AIS system.

Figure 1



AIS FLOWCHART
SOURCE DOCUMENTS
APL DIABLO 1550
IP SHARP NETWORK
AIS DATA BASE (APL FILES)
TELENET FILE
INFOREX TRANSMISSION
TELENET FEED
TELENET OPERATIONS DATA DISSEMINATION
TICKET NETWORK
BRANCH MESSAGES
AIS REPORTS
INQUIRIES
SEARCHES
FOR EACH REGION DP CENTER
ATF FILE
INFOREX TRANSMISSION
REGION ATF FEED
REGION ID CARD
REGION COBOL PROGRAM
ATF FEED
ATF DATA BASE
ATF REPORTS

Thus, a single entry of a batch of serial numbers: (1) immediately updates the AIS data base; (2) provides a digital feed to the batch equipment accounting system; and (3) provides the indicated Telenet nodes with a copy of the input batch. In addition, the Input program is designed to minimize the necessary number of keystrokes for data entry. This is done by combining a judicious choice of defaults (different for each ELC) with a changed field carry-down feature. The SHARP APL System's ability to dynamically invoke packages allows us to change the details of programs (e.g., sublocation names) and global variables to better suit each user while maintaining a single system.

As a batch of serial numbers is entered the Input program performs numerous edit checks on the data. When an error is encountered, the program prompts with a diagnostic message. This allows immediate correction of bad data. In addition, since the feeds to the equipment accounting system and to the Telenet network are not appended to the file until the editing process is completed, syntactic input errors are virtually eliminated in the data feed from the AIS system. Reports reflecting the updated information are immediately available.

When machines are shipped from an ELC, the logical records of those machines are not deleted from the AIS system. Likewise when an ELC chooses to delete or alter records in its data base (via the Delete or Change program) they are not physically deleted from the AIS system. Rather the records are moved to a separate Shipped or Delete sublocation. Thus, when an ELC needs to research information on a machine not currently in inventory, it can do so by referencing the special Shipped or Delete sublocation in the Inquiry program and that sublocation will be searched. This provides an ELC with an audit trail on each serial number that was ever entered into the system. Periodically this inactive data is dumped to tape.

A feature of the AIS system which has proved useful is the ability to allow selected individuals or groups concerned with regional inventory control to have read-only access to an ELC's data base. They can then run any of the report functions, effectively reducing the requirements placed on the ELC for information production. This is a major advantage since these individuals and groups tend to be geographically separated from the ELC's they control. Thus, the APL system provides an important report dissemination function. Finally the ability to rapidly develop enhancements and new application programs and activate them simultaneously throughout the country is a major advantage of an APL system. It allows a relatively rapid systems response to changing business requirements.

Problems

All of the systematic problems with the AIS system can be traced to a single factor, use. On average, the system is used over 800 connect hours a month for both entering data and printing reports. This gives rise to two major problems. First is cost. Significant portions of the computer's resources are used by AIS, calling into question the cost benefit ratio of many of the on-line services the system provides. The very ease with which reports can be requested encourages the functional user to request more reports on-line than may be cost effective. Since individuals in the field make these requests, central (cost) control is difficult.

Second, whenever any system problems occur such as the lines going down or a system crash, someone is using the AIS system. When the system is down for even a few hours, clerical personnel at the ELC are unable to input their daily data and must queue the input documents until the system comes back up.

The users of the AIS system are even affected by problems which occur when they are not on the system. The system creates over 130 tape feeds in a given month, each of which must be transmitted over telecommunication links. Both computer and telecommunication failures (which happen at night when these feeds are being created) adversely impact ELC operations (i.e., data feeds are late).

A potential Achilles heel of the AIS system is ironically due to its automation of the feed process. Handling 130 tapes a month is a manual procedure with ample potential for error.

Summary

The feature which distinguishes AIS from most of the APL programs I have encountered is its use as a large business system fully integrated with other non-APL systems. This type of use of APL is significantly different from its more common role as powerful language for expressing and running interactive mathematical models. In contrast, the AIS system relies heavily on the SHARP APL systems data communications facilities to allow a central computer to be the custodian of the physical file system and programs while allowing each field location to maintain a logical data base. Interactive data entry, a flexible file subsystem, and a logically consistent method for manipulating character arrays combine to make APL a useful and productive language for business systems development.

APPLICATION OF APL MODELING IN A PETROLEUM DISTRIBUTION NETWORK

Edmund C. Payne
Arthur D. Little, Inc.
Cambridge, Massachusetts

As part of a larger APL-based computer model of a petroleum distribution system, a module was designed to determine the lowest cost distribution network for a given set of bulk plants, refineries, demand centers and transportation links. Demand data was fed to the programs by a demand forecasting module and the distribution analysis in turn fed a cash flow system projecting investment requirements. The implementation was done on an IBM 5100 minicomputer supporting APL with a maximum workspace of 64K -- a somewhat limiting factor which required breaking the distribution model down into several smaller sections communicating with each other via APL data files. Even with such limitations, the minicomputer was chosen for its ease and simplicity of operation (standalone capability), reliability in harsh conditions, and low cost for development work. The application is sophisticated in its interaction with a fairly large data base and, for that reason, an interesting experiment in the application of APL on small minicomputers to large scale modeling efforts.

One of the major reasons for using APL for this model was the need for interactive capability combined with complex array manipulations. It was particularly important to optimize CPU time with such a slow machine (the final program ran for almost three hours) yet, at the same time, meet a very rigorous completion schedule. As it turned out, the interactive capabilities of the model became a necessity when inaccuracies in the data were discovered. In addition, the actual users of the model were unsophisticated at interfacing with computers and were greatly benefited by the ease with which they could quickly acquire a working knowledge of APL -- at least to the extent of being able to manipulate and change the data base and perform simple analyses on the model's results.

The major constraint of using the IBM 5100 was the limitation to a heuristic logic, creating problems in constraining the system flows and acquiring a greater degree of user participation (though due to the problems with the data, this was not necessarily a serious handicap). Using this type of methodology, it is impossible to guarantee that the optimum routing has been found -- the use of stepwise iterations can effect the choice of a route where the cumulative cost is greater but the link being currently examined has a lower cost than its immediate alternatives.

The model consists of two parts: the interactive programs which allow the user to define the network configuration, and the actual core programs which calculate the least cost routes. The model allows changes to all data stored in the files; the year of interest in the ten-year planning horizon, refineries (Table 1), pipelines configurations (Table 2), with diameter options of 8″, 10″, 12″ or 16″ bulk plants and the ability to reverse or drop transportation links (Table 3) -- this scheme allowed for placing refineries in what would normally be demand centers; reversing the flows would make the site a supplier instead of a user. Once the network options have been defined, a table containing two hundred and fifty possible transportation links is read in from file and edited by the program to reflect modifications made by the user in the previous step. All links which cannot participate in the network due to removal of intermediate bulk plant are removed. At this time, duplicate routes are dropped on the basis of calculated unit costs. There are a variety of "types" of duplicates and a series of logical "rules" (in addition to costs) for removing them. For example, pipelines take precedence over all other routes, as do bulk plants directly serving their demand centers. In other cases, the links which are available depend on the year being examined -- many road and rail links are in the planning stage only, with projected completion dates in the early 1980's. The final result of this group of programs is to produce a table of unique links with associated unit costs (per barrel) and demand center volumes for the seven fuel types being examined: premium and regular gas, jet fuels, diesel and fuel oil.

TABLE 1

LIST OF REFINERIES

```
*IM
*JD
*JZ
*RT
*RI
*YN
*BD
```

TABLE 2

NEW PIPELINE LINKS (FILE: 'NEWPIPE')

```
PIPE    1  1
PIPE    2  2
PIPE    3  3
PIPE    4  1
PIPE    5  2
PIPE    6  4
PIPE    7  1
PIPE    8  2
PIPE    9  1
PIPE   10  5
PIPE   11  1
PIPE   12  2
PIPE   13  1
PIPE   14  1
PIPE   15  1
PIPE   16  1
PIPE   17  1
PIPE   18  1
PIPE   19  1


       GO
PIPE : 1
 1  DH RI N10177450
PIPE : 2
 1  RI QA N20177 443
 2  QA HA N20277305
PIPE   3
 1  DU TA N30177 185
 2  TA JO N30277355
 3  JO TU N30377451
PIPE : 4
 1  YA ME N40177246
PIPE : 5
 1  JE MA N5017772
 2  MA TA N5027788
PIPE : 6
 1  RI KH N60177  64
 2  KH LA N60277 215
 3  LA SU N60377241
 4  SU NA N60477 434
PIPE : 7
 1  KH SU N70177 539
PIPE : 8
 1  JI KM N80177237
 2  KM NA N80277264
PIPE : 9
 1  JI NA N90177435
```

```
PIPE : 10
 1  *RTJU NA017784
 2  JU NA NA027799
 3  NA QA NA0377251
 4  QA BA NA0477580
 5  BA TU NA0577238
PIPE : 11
 1  BA JO NB0177213
PIPE : 12
 1  *RTJU NC017784
 2  JU SA NC0277 105
PIPE : 13
 1  *RTHO ND0177 165
PIPE : 14
 1  *RTQA NE0177887
PIPE : 15
 1  RI NR NF0177  66
PIPE : 16
 1  JE NO NG0177  41
PIPE : 17
 1  DH DA NH0177   8
PIPE : 18
 1  *RIRI NI01770
PIPE : 19
 1  JE YA NJ0177358
```

TABLE 3

LINKS TABLE (FILE: LINKS)

```
 1  *BD BA   T500 77    0        49  DH  RI   T158 77  450
 2  *IM DU   M309 77 2.50        50  DH  SA   T008 77  269
 3  *IM JE   M186 77 2.50        51  DH  SU   T313 77 1006
 4  *IM JI   M302 77 2.50        52  DH  TU   T010 77 1280
 5  *IM QU   M304 77 2.50        53  DH  01   Z350 77 154
 6  *IM RA   M307 77 2.50        54  DH  02   U02  77    0
 7  *IM SA   M306 77 2.50        55  DH  021  Z349 77 30
 8  *IM YA   M300 77 2.50        56  DH  023  Z194 77 123
 9  *JD JE   T080 77    0        57  DH  03   Z351 77 269
10  *JD NO   T185 77 41          58  DH  04   Z198 77 462
11  *JZ JI   T177 77    0        59  DH  05   Z352 77 450
12  *RI OR   T157 77   15        60  DH  06   Z354 77 887
13  *RI RI   T120 77    0        61  DH  08   Z355 77 1042
14  *RT DH   P001 80   37        62  DH  09   Z356 77 1280
15  *RT DU   M020 77 4.34        63  DH  20   Z334 77 1160
16  *RT JE   M083 77 2.39        64  DH  21   Z335 77 1006
17  *RT JI   M303 77 3.67        65  DH  22   Z359 77 527
18  *RT JU   M046 77 .9          66  DU  JO   T023 85  495
19  *RT QU   M305 77 3.83        67  DU  TA   T030 85  140
20  *RT RA   M308 77 4.01        68  DU  11   U111 77 0
21  *RT SA   M036 77  .99        69  HA  BA   T503 83  684
22  *RT YA   M301 77 2.88        70  HA  RA   T108 83  400
23  *RT 022  U022 77    0        71  HA  07   U07  77    0
24  *YN YA   T169 77    0        72  HO  01   U01  77    0
25  BA  TU   T501 77  238        73  JE  150  U150 77    0
26  BA  07   Z666 83 684         74  JE  BI   T097 80  603
27  BA  08   U08  77    0        75  JE  DU   M019 77 3.65
28  BI  16   U16  77    0        76  JE  JE   T400 77 1000
29  DA  020  U020 77    0        77  JE  JI   M081 77 3.85
30  DH  BA   T106 77 1042        78  JE  JO   T026 77 1195
31  DH  DA   T001 77 9           79  JE  KM   T049 78  757
32  DH  HO   T062 77  154        80  JE  MA   T076 77   72
33  DH  HO   R063 77 9000        81  JE  ME   T321 77  424
34  DH  JO   T028 77 1255        82  JE  QU   T092 80  408
35  DH  JU   T130 77  123        83  JE  QU   M171 77 3.18
36  DH  KH   T005 77  527        84  JE  RA   M039 77  .94
37  DH  KH   R006 77 9000        85  JE  RA   T040 77  154
38  DH  LA   T314 77  749        86  JE  TA   T035 77 1110
39  DH  NA   T312 83 1160        87  JE  TA   T323 77  160
40  DH  NA   T203 77  211        88  JE  UL   T094 78  520
41  DH  NR   T311 77  450        89  JE  YA   M044 77  .94
42  DH  OR   T078 77  450        90  JE  YA   T045 77  358
43  DH  OR   R079 77 9000        91  JE  ZA   T085 77 402
44  DH  QA   T118 77  462        92  JE  10   Z329 77 1195
45  DH  QA   T003 77  887        93  JE  11   Z328 77 1110
46  DH  QA   T181 77   30        94  JE  111  Z328 77 840
47  DH  RA   T110 77  758        95  JE  12   Z171 77 424
48  DH  RI   R133 77 9000        96  JE  13   Z165 77 358
```

All the data needed for calculation of the optimum routings is now available. Beginning with the refineries, a cost build-up and alternative cost comparison of the network is initiated; in the first pass only bulk plants connected directly to refineries are costed. The rule for movement through the network is that only destinations with all origins already assigned built-up costs can themselves be costed. Thus, if three different routes lead to an unpriced bulk plant cost comparisons are made only if the origins of all three routes have been given a built-up cost. The program continues through successive iterations until all demand centers have been reached and a total route cost is known. It is at this point that a weakness of this approach can be observed -- the program is only looking at built-up costs, one step at a time. There is the possibility that there would actually be a more optimum route formed by picking a longer route in the first pass to get an even shorter link in the next pass. The number of iterations was not great -- ranging between four and seven -- and on an overview of the system allows for identification of blatant discrepancies (this overview occurs as a printout of chosen links with associated built-up costs). When this stage finishes, all demand centers have been assigned a total unit cost and a complete set of transportation links to a refinery. The program now retraces the route from demand center to refinery -- now a recursive looping over refineries, one at a time. This is relatively simple -- note that while a bulk plant can supply a number of plants further down the line (origin to destination), there is a unique origin for any bulk plant supply. That is to say, there is a unique destination-to-origin transportation link for every bulk plant -- the elimination of multiple suppliers is, afterall, what the calculations in the first step accomplished.

As the program backtracks to the refinery, a running tally is kept for each bulk plant on the seven products flowing through it. This occurs for each demand center and eventually yields a total flow through each bulk plant and a total demand on each refinery. A simplified view of the process is the building of a triangle with each refinery at the apex; as you move towards the top, volume throughputs successively increase until the top shows the full demand for all supplied centers. The flows may be double counted as many as seven times. In addition, an actual total cost is stored for each transportation link (unit cost times flow from each demand center) along with the total flow. The resulting data base can be drawn on for a number of valuable reports.

Having calculated the total required storage for each bulk plant, the next step is to use inventory level algorithms which calculate the total required storage to efficiently operate the system with an adequate safety margin (Table 4). Comparison of these requirements to existing storage (Table 5) identifies a plant as either a grass roots or expansion project. The program picks up the appropriate investment curve from stored tables (Figures 1 and 2) and returns the total investment required for each bulk plant. In the same way, pipeline project investments are found.

Utilizing transportation link data stores in previous sections, total volume traveling over highway routes can be broken down into quantity of trucks needed; downtime, maintenance, and scheduling are all subtracted from the available pool to give the number of additional trucks needed and the associated investment incurred.

The same process in now followed for fixed and variable operating costs (Figure 3). As with investment costs, appropriate values are looked up on stored unit cost curves -- in actuality, a two-row matrix representing points on the curve, linearly interpolated. More accurate costs can now be assigned to the refinery-demand center linkage giving accurate volumes with which to calculate operating costs (this is another of the disadvantages of the heuristic approach -- absolutely accurate cost comparisons of routes cannot be made earlier in the model since the total flows for the network have not yet been arrived at and bulk plant handling charges must rely on unit costs with no account being taken for bulk load discounts).

TABLE 4

MAXIMUM DEMAND-RELATED STORAGE REQUIREMENTS

Mode	Lead Time (days)	Frequency of Supply (days)	Max. Inventory (days supply)
Truck*	6	1	5.4
Truck	4	1	4.6
Truck	2	1	3.6
Rail	5	4	8.0
Rail	5	2	6.0
Rail	3	4	7.1
Rail	3	2	5.1
Pipeline	5	2	6.0
Pipeline	3	2	5.1
Pipeline	5	4	8.0
Pipeline	3	4	7.1
Barge+	18	2	9.7
Barge	12	2	8.3
Tanker++	18	3.6	11.3
Tanker	15	3.6	10.6
Tanker	12	3.6	9.9

*358.5 barrels per semi-trailer
+25,000 barrels
++150,000 barrels

TABLE 5

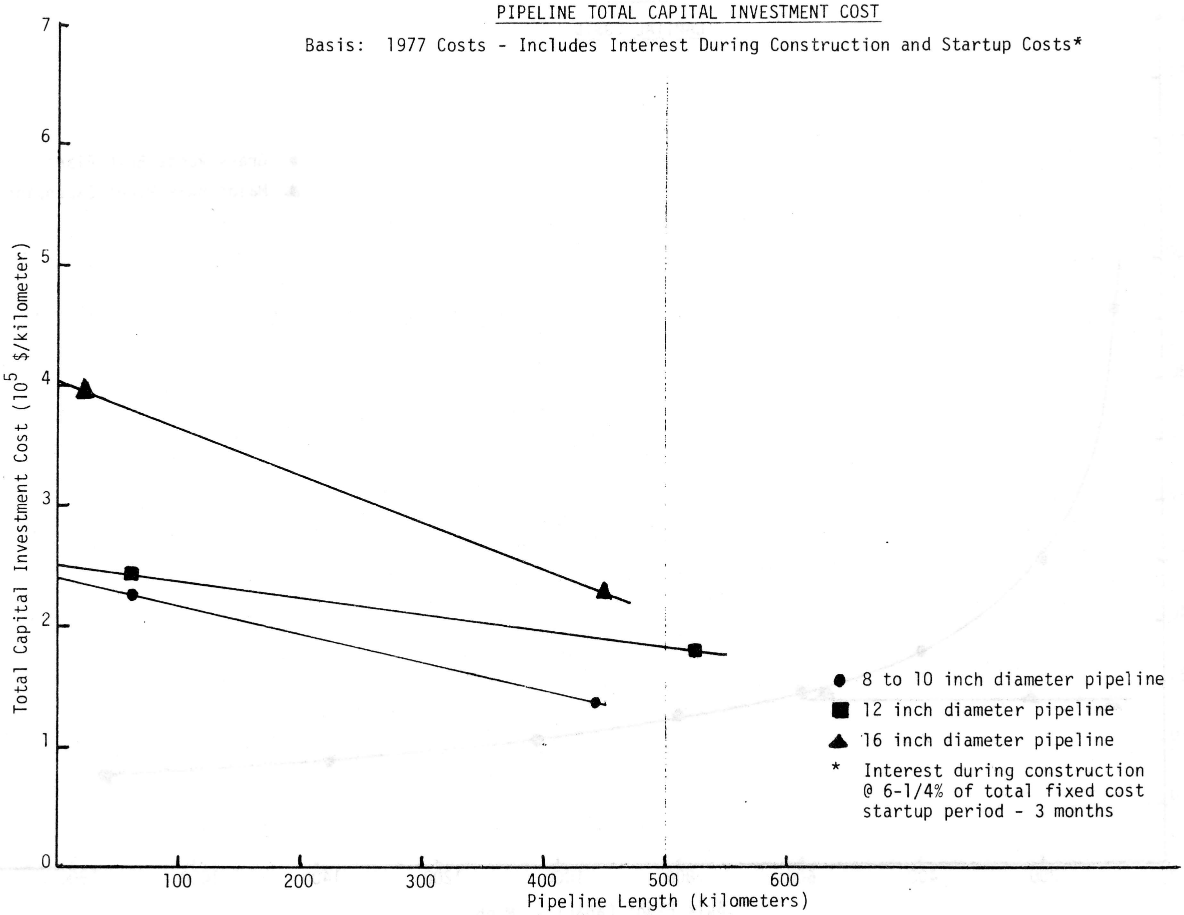
BULK PLANT STORAGE CAPACITIES – EXISTING (1/1/78) AND FUNDED AND/OR UNDER CONSTRUCTION

Location		Premium	Regular	Total	Avgas	A-1	Kerosene	Diesel	Fuel Oil	Marine Diesel	Total
Dh	Existing	78	54	132	15	116	58	340	–	–	661
	Planned	300	–	300	–	–	–	900	–	–	1,200
	Total	378	54	432	15	116	58	1,240	–	–	1,861
Du	Planned	60	60	120	–	–	60	60	30	–	270
Ho	Existing	3	3	6	–	–	1	3	–	–	10
	Planned	5	–	5	–	–	–	10	–	–	15
	Total	8	3	11	–	–	1	13	–	–	25
Je	Existing	204	292	496	56	476	–	1,329	517	–	2,874
	Planned	46	(42)	(496)	(56)	(476)	–	(1,329)	(517)	–	(2,374)
	Total	250	250	–	–	–	–	–	–	–	500
Jiz	Existing	60	30	90	–	30	60	60	–	–	240
Jo	Planned	120	20	140	–	–	10	150	–	–	300
Kh	Existing	21	3	24	–	–	2	24	–	–	50
Na	Existing	20	160	180	–	–	3	80	–	–	263
Qa	Existing	3	40	43	–	–	3	40	–	–	86
	Planned	20	–	20	–	–	–	20	–	–	40
	Total	23	40	63	–	–	3	60	–	–	126
Ri	Existing	20	4	24	–	–	2	16	–	–	42
Ri	Existing					Not Available					472
	Planned										948
	Total	400	400	800	20	200	–	400	–	–	1,420
Sa	Existing	10	5	15	–	–	4	10	–	–	29
	Planned	20	–	20	–	–	–	–	–	–	20
	Total	30	5	35	–	–	4	10	–	–	49
As	Existing	10	40	50	–	–	3	30	–	–	83
Ta	Planned	60	60	120	–	–	30	120	–	–	270
Tu	Existing	5	–	5	–	–	1	4	–	–	10
	Planned	20	–	20	–	–	–	20	–	–	40
	Total	25	–	25	–	–	1	24	–	–	50

FIGURE 1

PIPELINE TOTAL CAPITAL INVESTMENT COST

Basis: 1977 Costs - Includes Interest During Construction and Startup Costs*



* Interest during construction @ 6-1/4% of total fixed cost startup period - 3 months

FIGURE 2

CAPITAL COSTS

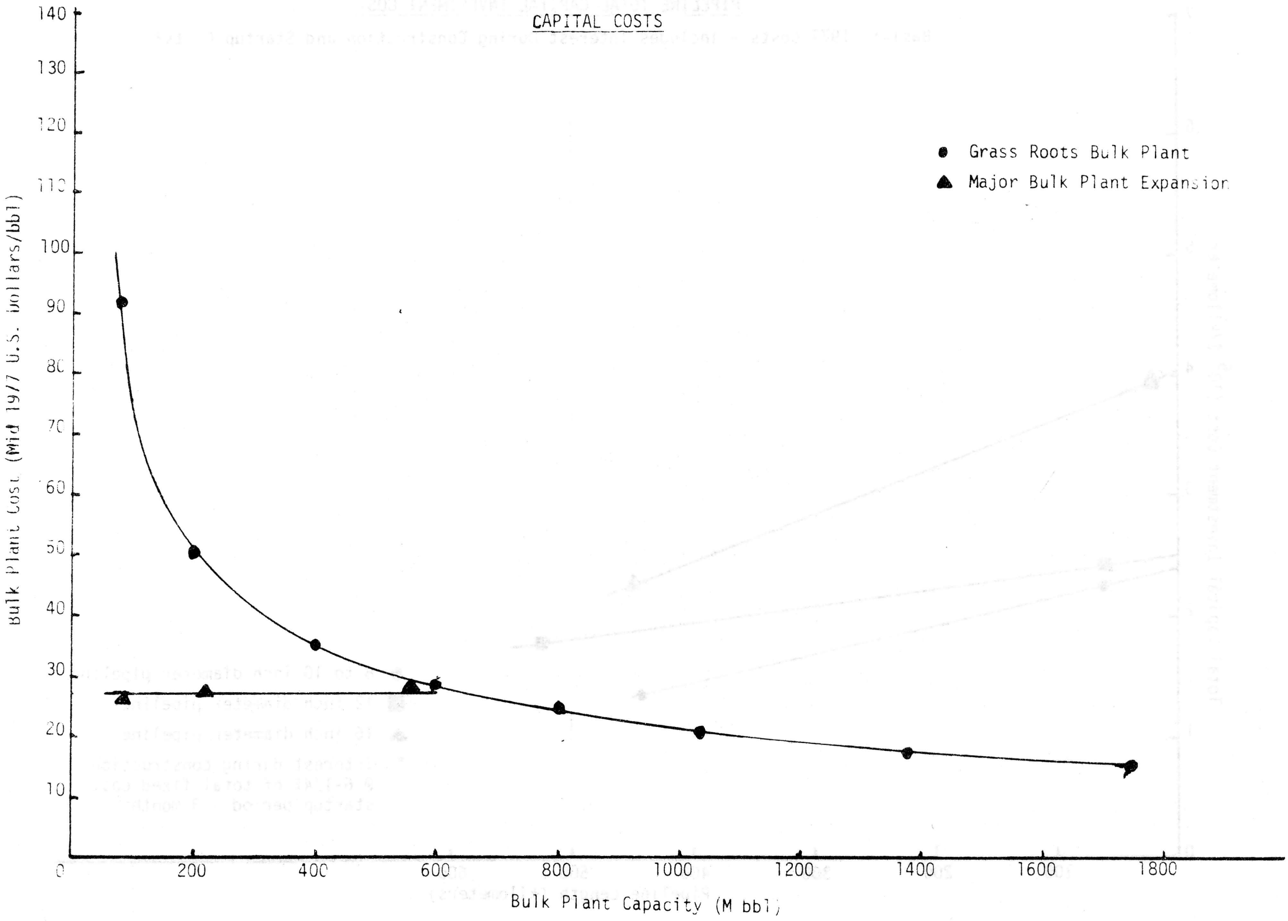
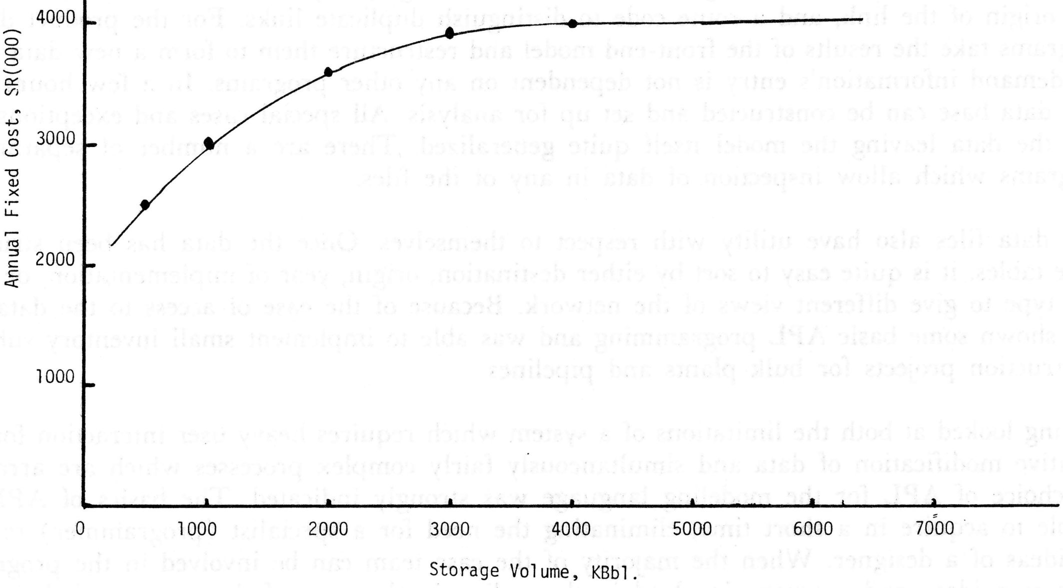


FIGURE 3

BULK PLANT FIXED COST

Basis: 1977 - Excludes Depreciation



Finally, total demand on refineries is compared to available output and the amount of imports or increased refinery capacity required are established. Since the system cannot be constrained, to adjust refinery production, certain transportation links are removed in some scenarios to force certain demand centers to draw on projected new refineries.

The model is extremely flexible in its capability of being adapted to a wide range of different problems. The programs themselves are quite generalized and it is the characteristics of the data base that determine the application. Through a series of external programs, the data files can be readily adapted to various network costing problems. Definition of the transportation links table is the key to setting up a new system -- this table contains the type of linkage (truck, rail, barge or pipe), the date the link will be in service, the mileage (or, in the case of barges, ship size and route cost), the destination and origin of the link, and a route code to distinguish duplicate links. For the product demands the programs take the results of the front-end model and restructure them to form a new data base. Thus, the demand information's entry is not dependent on any other programs. In a few hours an entirely new data base can be constructed and set up for analysis. All special cases and exceptions were built into the data leaving the model itself quite generalized. There are a number of separate reporting programs which allow inspection of data in any of the files.

The data files also have utility with respect to themselves. Once the data has been structured into these tables, it is quite easy to sort by either destination, origin, year of implementation, or transportation type to give different views of the network. Because of the ease of access to the data, the client was shown some basic APL programming and was able to implement small inventory subsystems on construction projects for bulk plants and pipelines.

Having looked at both the limitations of a system which requires heavy user interaction for somewhat intuitive modification of data and simultaneously fairly complex processes which are array oriented, the choice of APL for the modeling language was strongly indicated. The basics of APL are quite simple to acquire in a short time, eliminating the need for a specialist (programmer) to implement the ideas of a designer. When the majority of the case team can be involved in the programming of their own ideas and everyone involved in the coding is also part of the conceptual design team, an incredibly efficient and rapid model-building effort is created. APL's ability to act as a mathematical shorthand in addition to being an efficient programming language make it both more acceptable and more readily acquired as a skill by professionals in widely diversified fields. This flexibility and broad appeal as well as the results of many models, have been responsible for the acceptance by the users, since a better understanding of the actual internal processes removes the black box stigma generated by uninvolvement of the designers in the actual implementation. Only through the use of a mathematically concise language such as APL can applications of this complexity be attempted when time and man hour (professional cost) budget considerations are fairly restrictive.

FINANCIAL MODELLING: KEYS TO SUCCESSFUL IMPLEMENTATION

Gary J. Beagle
Canadian Occidental Petroleum Ltd.
Calgary, Alberta

Financial Modelling In The Resource Industry

Introduction

The Resource Industry, specifically the Oil & Gas sector, has grown significantly over the past twenty years, with the most dramatic increase (revenue and investment) occurring in the past two to three years. Other sectors such as minerals, forestry, etc. have not increased nearly as rapidly. In fact, the minerals sector is declining due to depressed product prices and excessive taxation. In the recent past, lumber prices have moved up and as a result, new investment opportunities are apparent in this sector.

The purpose of this paper is to discuss the magnitude of the Oil & Gas sector and why financial modelling is a **must** in order to survive over the long term. The paper will also discuss how to implement a financial model within the organization and conclude by discussing some potential extensions and further adaptations.

The Oil & Gas Sector

In order to discuss the merits of financial modelling, we should first discuss the magnitude of this industry, to give a perspective of the dollars and cents involved. This sector is generally divided into three major "functions": (1) Exploration and Production, (2) Transportation, and (3) Refining and Marketing. It is important to discuss these functions separately because, by their nature (and by their profitabilities), they are truly different businesses.

The "Transportation" function operates primarily as a utility-type operation, specifically with reference to pipelines. This is generally a result of their "semi-monopoly" status, with the tariffs usually based on cost of service (cost plus return on invested capital) which is approved by the necessary regulatory agencies. Usually, a consortia of companies are set up to finance major projects (many of whom have an interest in the product actually being transported). Competition for major pipeline investments (Montreal Maritimes P/L, Alaska Gas P/L, Sarnia-Montreal P/L, etc.) at best consists of two to three consortia.

Refining (including petrochemicals) and marketing operations relate more to the manufacturing and consumer type industries. Refining is an oligopolistic market with a finite number of competitors. A combination of shift in product demand from motor gasoline to middle distillates, conservation, reduced demand as a result of increasing prices and the construction of substantially more refining capacity in central and eastern Canada, have resulted in significant overcapacity. Crude oil refineries will be operating at about 65% capacity by the end of 1978, including Petrosar's Sarnia refinery and Texaco's new Nanticoke refinery.

Similarly, the marketing side of operations has had to deal with heavy competition. As the price of gasoline rises, consumers begin to shop around to save dollars. As a result, independent discounters have entered the market, forcing the majors to convert to self-serve operations and consolidate by "purifying" their network (eliminating unprofitable operations).

Because of poor market conditions, major expansionary projects in the refining and marketing function have been curbed with the exception of maintenance, cost efficiency, pollution abatement type projects. This is not to say that financial modelling does not have applications in this area, as it is very much necessary to survive through efficiencies and optimum pricing policies.

As the threat of a world energy shortage looms (and ebbs), so does the extensive (and expensive) search for both conventional and non-conventional sources of energy. Product prices for crude oil and natural gas are effectively regulated by the National Energy Board. In 1973, the OPEC countries unilaterally increased crude prices, changing the economic situation of the entire western world. Oil prices escalated rapidly over the last five years, from $2.90/Bbl to a current average field price of $12.75/Bbl, initiating conservation and attracting substantial new investment. Similarly, gas prices have risen approximately tenfold over this same period. In 1977, gross revenue from oil and gas production in Alberta alone, exceeded six billion dollars. As a result of the increased cash flow, investment levels have reached all time highs resulting in intensive competition within the industry. The Canadian Petroleum Association recently reported industry capital expenditures including oil sands, of $7.6 billion in 1977. In excess of 6,000 oil and gas exploration and development wells were drilled in Canada. Drilling rigs and service companies are effectively operating at 100% capacity. Total bonuses paid at government land sales amounted to in excess of $700 million in 1977 (up from just over $200 million in 1976).

No longer can one rely on manual calculations to evaluate all the potential investment alternatives and the associated ramifications facing the industry today. Diverse royalty and taxation structures further complicate evaluations. In order to survive and grow in this environment and to comprehend the major factors affecting decision-making, companies must employ increasingly sophisticated management tools.

Why The Need For A Corporate Financial Model?

First of all, we should define the corporate financial model. The model generally is a financial projection and economic evaluation of a project or group of projects. Operating parameters such as production, sales, prices, operating costs, taxes, capital and working capital requirements, etc. are estimated over the expected life of the project in order to calculate the project's viability. A corporate financial model must accomplish these calculations, and in addition have the ability to combine several new projects with existing operations to ascertain the viability of the entire company.

When we commenced the development of a corporate financial model at Canadian Occidental Petroleum Ltd., we established the following objectives:

(1) To provide a mechanism to develop long-term pro forma financial projections.

(2) To provide a mechanism to evaluate the impact of major projects and acquisitions on the company's overall financial forecast.

(3) To provide a means for identifying corporate financing abilities to support major projects or acquisitions, including projection of cash surplus or cash requirement.

(4) To provide a mechanism to identify the "critical performance areas" and key factors which will have the most effect on the company's future in terms of financial results.

(5) To provide a tool for ranking and/or phasing major projects or acquisitions from an overall corporate point of view.

(6) To provide a tool for fast, efficient, and consistent assessment of the "what if" questions arising from alternate premises such as government legislation, expansions, divestitures, etc.

Excessive effort is required to accomplish the above objectives on a manual basis. In fact, it would be impossible to do so in today's business environment. With manual calculation, long lead times are required, flexibility is limited, and clerical errors and inconsistent application negate the results of the output. For example, once the economic premises have been established and all relevant calculations associated with a financial projection have been completed, some new development (or idea may require a new price forecast. As a result, the number crunching must be repeated in total. This same situation can occur time and again, considering the multitude of variables incorporated in any corporate financial projection.

Once the decision has been made for the need of a computerized corporate financial model, the next step is to develop a plan for implementation. As with any plan, a certain amount of flexibility must be incorporated. Because of changing needs, priorities and changes in corporate structure, the specifications will undoubtedly be altered during the implementation phase.

Selling the Corporate Model

Developing a "tailor made" corporate financial model is an expensive proposition. Most of us have heard about the classic Sun Oil model developed a number of years ago which required many man-years of effort! This leads to an interesting point regarding a compromise between accuracy and practicality, which will be discussed in more detail later. In order to get a commitment on a project of this nature, particularly since it involves the entire organization, top management approval must be obtained. Ideally, the Chief Operating Officer should be sold on the project and thus ensure support of the project by senior staff members. The first step is to display the merits of a computerized financial forecast system, including an explanation of how this management tool will assist in the decision-making process for the entire corporation. This can best be emphasized by illustrating how a model could have provided significant assistance in a recent major investment decision. The illustration will show how the corporate financial picture would have looked in the future with and without the project, under various volume, price and cost assumptions.

The proposal and initial plan of action should be properly documented in a report format. Included should be such basic items as project description, purpose, justification, manpower requirements, completion dates (by segment), etc. Estimates on completion dates and total cost should be realistic. Our model took over a year to complete from inception to ultimate utilization. This is substantially above the original estimate for a number of reasons, most of which were unforeseeable at the time.

Top level approval for the project will have other beneficial side effects. Senior management will want progress reports on a regular basis. As opportunities to utilize the model become apparent, pressure will increase at all levels of the organization for progress in its implementation.

Who Should Be Involved

It is beneficial to establish a "committee" made up of key personnel from each of the various operating departments within the company, including the accounting and budgetting groups. These members should be appointed by senior management and be made responsible for ensuring that model specifications accurately represent their respective departments' operation.

The next step in the development process is to work with each department representative, on an individual basis, to obtain a detailed knowledge of how and what a financial model could do for them.

Generally, most corporate financial projections are done by the accounting department. In order to obtain their support and acceptance, they will have to be convinced that the model will duplicate their manual calculations. This requirement will add significantly to the size and complexity of the model. However, compromises will have to be made in order to minimize the cost and input requirements for the model. As previously mentioned, decisions will be required trading off accuracy against practicality/cost.

These decisions will have a large impact on acceptance and use of the Corporate Model. Examples of where compromises must be made are exemplified in royalty calculations. Ideally, royalties should be calculated on a tract by tract or lease basis; in the case of crude oil, on a well by well basis. However, this is totally impractical, particularly when modelling a company's existing production. At CanadianOxy, we developed production forecasts for each product on a province by province basis. Production is then allocated to "new" Crown and Freehold on a percentage basis (input as a variable) with the remainder being assumed "old" Crown production for calculation of royalty/mineral tax. Once these percentage splits for the company are accurately established, then the calculations become reasonably representative. When a particular project is being evaluated separately, the royalty interests are known and can be input on an exact basis.

Another example of where approximation becomes useful is in the calculation of provincial income taxes. Because provinces have different tax rates and thus provincial tax calculations vary significantly for resource income, it is more practical to use percentage allocation of income to "producing" and "non-producing" provinces. Actual provincial allocation is based on the combined ratio of sales revenue and salaries, not on the basis of actual income earned. This calculation would be difficult to model for each case, and a simple allocation between producing and non-producing provinces, using respective current average tax rates and current actual allocations, is much simpler and sufficiently accurate.

Innumerable instances arise in the development of the specifications, requiring similar judgement on the most practical approach for developing an algorithm. These decisions will depend to a large degree on what aspects the specific company feels are relevant to their operation and planning.

Another area where decisions on practicality will be encountered, will be the conversion of specific departmental requirements into one common set of generalized specifications. At this point, it will be necessary to call all representatives together in order to obtain agreement on the type of calculations and, equally as important, the type and format of output.

In summary, all operating and financial departments should be involved in the development of the corporate model. As a financial planner, it will be necessary to coordinate and control, or in other words, lead the project to completion, and yet not be constrained by the typical "committee" environment.

Technical Programming Support

Once we have agreed on the specifications, we are now in a position to solicit the technical support required to carry out the implementation of the model. The ultimate usefulness and flexibility of the model will largely be determined by the quality and type of technical support used. Whether this support comes from inside or outside the firm, the people involved must not only be proficient programmers/analysts, but also be knowledgeable to some extent in basic accounting. This will not only minimize explanations to the programmer, but will also reduce corrections and revisions. The programmer will then be able to spot inconsistencies and develop a logical progression of routines within the main program. Routines or functions should typically be made up of ten to fifteen lines of code. This will allow for ease and flexibility in programming changes.

Input Flexibility

Regardless of how detailed the specifications, or how many alternatives are built into the input format, there will always be instances when this is inadequate for a specific project. Changes in income tax regulations are a frequent example.

It is therefore necessary to provide for flexibility of data input methods. Where percentages are input for certain calculations, it helps to provide for manual data input to override those calculations. It is also useful to have the ability to manually schedule other data (both taxable and non-taxable). By incorporating flexibility in the method of inputting data, practically any type of project can be included in the model in one fashion or another.

Output Reports

Typically, output reports from the Corporate Model will be used by a wide range of people and by several levels of management within the organization. Some people are of the opinion that output reports need merely provide basic data which can then be transposed manually to a presentation format. This is a time-consuming effort which also leads to errors. Output reports should thus facilitate direct presentation to the ultimate user.

Since users of corporate model information will have different backgrounds, it is important to adopt a standard report format, easily understood by all. The accounting (financial) format has become standard and is used in most management reports. Thus, it is useful to adopt a similar style for the Corporate Model. The executive is already familiar with the format, and the numbers become more meaningful and are easily comparable to historic data.

At Canadian Occidental, we presently have four output reports (pages) for each project/division, with the exception of oil and gas which has five. The first page is a summary of the production/sales volumes and prices by product; percentage increases (calculated) are also shown along with total revenue by product and in summary. The second page, applicable only to oil and gas projects, summarizes royalty and mineral tax payments by product (crude oil, natural gas, etc.). The third page is a basic income statement. On all statements, time in years is formatted horizontally and financial elements are listed in vertical order, with appropriate headings. The income statement, therefore, is very similar to one featured in an annual report. The fourth page is the cash flow statement, basically divided into sources and uses. Sources include net income, non-cash changes, new debt and/or financing, proceeds from disposals and other. Uses include capital and exploration expenditures, working capital increases (decreases), dividends, debt and equity redemptions, income tax installments, adjustments, etc. The fifth page is the balance sheet which includes all relevant balance sheet items, including the year-end cash balances. Attachments A, B, C, D and E illustrate the typical output reports for oil and gas projects.

Documentation

In order to have a system accessible to several departments or users, it is necessary to document virtually all aspects of the Model. This is particularly important when using APL, as the programs do become somewhat personalized and are not self-documenting. Preparation of a detailed manual ensures that the use and maintenance of the Model is not limited to one or two people. This provides for continuity of use and hopefully provides the company with a significant and enduring return on their programming investment.

"Off The Shelf" vs "Tailor Made" Models

Basically, there are two approaches to developing a corporate model. The most popular one is to use an existing modelling "package", some of which come with a number of canned routines. The other approach is to develop the model from scratch. The pros and cons of these two approaches can be argued, each having its own merits. The existing packages are a very useful way to get into modelling as some results can be obtained very quickly and specifically programmed functions can always be used at a later date (if applicable) in a tailor made model. The development of a tailor made version requires substantially more systems knowledge. However, they do give full control over the entire model and will not contain redundant capabilities.

Why APL?

APL is radically different from conventional programming languages and apparently difficult to understand by some programmers. Non-programmers often grasp the basic language much more easily. So why use APL as simply another language to understand and support?

Basically, financial planning involves the manipulation of large matrices of numbers. With the power of APL, common algorithms can be programmed without creating loops, resulting in the ability to handle a multitude of data manipulations with very little code.

At times, APL is criticized because the functions become personalized and thus difficult to read by others. However, if each segment of the program is limited to ten to twenty lines of code, then the component can simply be discarded and a new segment inserted if a change is required.

Results To Date At Canadian Occidental

Canadian Occidental Petroleum Ltd. is a diverse company involved in oil and gas exploration and production in Canada, the U.S. Gulf Coast, South America and the Far East. We are also a major electrochemical manufacturer (Hooker Chemicals) in the West with plants in North Vancouver, Nanaimo, Squamish and Brandon. In addition, we operate the Parker/Sel-Rex Metal-Finishing Division in Toronto and the Durez Plastics and Resins plant in Fort Erie. Although we have no mineral deposits on production, we have a minerals division headquartered in Toronto which is currently actively exploring a number of interesting prospects in uranium, gold, lead, zinc, etc.

Near the end of 1976, we made the decision that a computerized financial model was the only way to effectively monitor our existing businesses and assess new investment strategies. Model development was not without problems, most of which related to determining what language and which computer vendor to use. These problems were compounded by a lack of time and continuity, due to acquisition evaluations and staff turnover. Further, the diverse and international aspect of the company's operations required special functions for each different business area.

The first actual full-scale use of the model was made during the preparation and presentation of our 1978 Strategic Plan early this year (approximately fifteen months from project start). Senior management used the model output for their review of respective departmental long-range plans, and were thus able to assess the financial implications for each of the proposed strategies. The corporate summary provided a ready comparison, in each case, of net income and cash flow, financing requirements, cash surpluses, stock redemptions, dividends, and all other financial measures. Reception to the model was highly positive and the future of this program is assured.

For modelling purposes, the company is currently segmented into thirty-one different parts, each of which can be examined separately and consolidated into the appropriate subgroups. It is thus easy to identify and reconcile changes in individual projects and demonstrate their effect on the company as a whole. We are also now able to update our Strategic Plan on a quarterly basis.

Proposed Future Developments

At this point in time, we have completed the initial phase of corporate modelling. Although, we now calculate certain percentage changes in prices and volumes, we intend to expand the financial analysis to include the calculation of relevant financial and operating ratios.

Our parent company, Occidental Petroleum Corporation, has recently undertaken to develop a corporate model covering their entire worldwide operations, including all of their widely diversified businesses. We expect to be able to provide significant assistance in this regard, as a result of experience gained to date.

For modelling purposes, the company is currently segmented into thirty-one different parts, each of which can be examined separately and consolidated into the appropriate subgroups. It is thus easy to identify and reconcile changes in individual projects and demonstrate their effect on the company as a whole. Our parent company, Occidental Petroleum Corporation, has recently undertaken to develop a corporate model covering their entire worldwide operations, including all of their widely diversified businesses. We expect to be able to provide significant assistance in this regard, as a result of experience gained to date.

In our next phase, we will look at developing bar and line chart formats which will depict more clearly key numbers from the financial output. Once these charts are accepted by senior management, we will be in a position to consider the development of computerized plotting routines which would use the data directly from corporate model data files.

Summary

Corporate financial models are highly useful management tools in today's resource business environment, which is pressured by competition, restricted markets (oil and gas exports) and increasing costs. Computer applications in seismic processing, reservoir engineering and economic evaluations have snowballed in recent years. Indications are that this situation will continue for several years in the future.

The foregoing approach to developing a corporate model applies to many industries, and should prove useful to those considering a similar study. Patience and perseverance are the underlying keys to success.

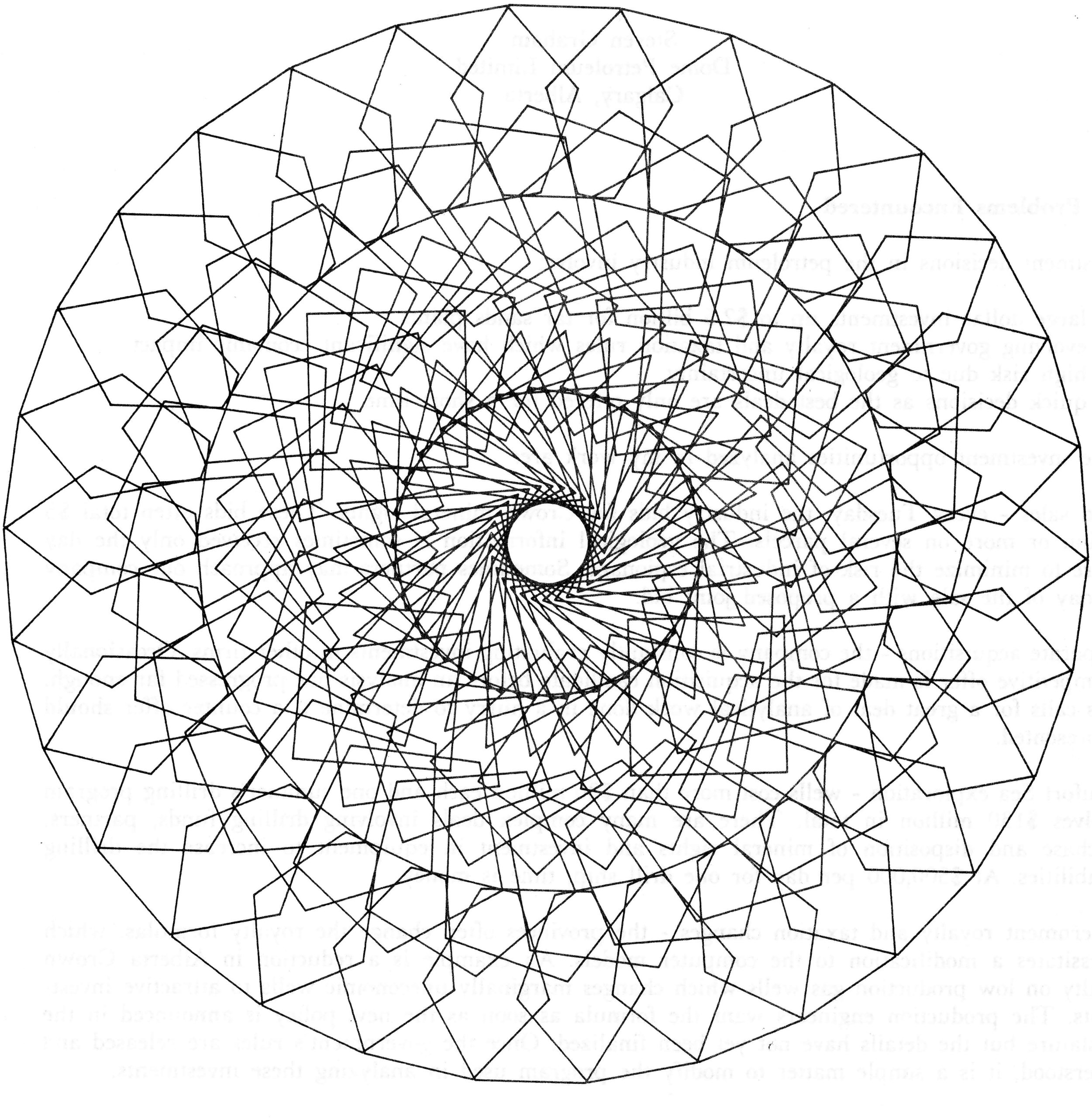
```
DIVISION:  DEVELOPMENT PROJECT EXAMPLE                           ATTACHMENT A
**************************************

                              CANADIAN OCCIDENTAL PETROLEUM LTD.
                                 CORPORATE FINANCIAL MODEL
                               PRODUCTION AND REVENUE SUMMARIES
                              **********************************

                                   1978    1979    1980    1981    1982    1983
       OIL AND GAS
      ============
      PRODUCTION VOLUMES-
      NET SALES BEFORE ROYALTY:
501       SALES GAS(BCF)            0.2     0.8     2.1     5.2     5.1     4.9
505       CONDENSATE(MSTB)          0.0     0.0     0.0     6.1     6.1     6.1
506       SULPHUR(MLTONS)           0.0     0.0     0.0    20.9    20.9    20.9
                                  -----   -----   -----   -----   -----   -----
      INCREASE/DECREASE(PCT.)
508       GAS                       0.0   327.8   169.0   152.5    ¯3.1    ¯2.5
510       CONDENSATE                0.0     0.0     0.0  9999.0     0.0     0.0
511       SULPHUR                   0.0     0.0     0.0  9999.0     0.0     0.0
                                  -----   -----   -----   -----   -----   -----
      PRICES:
191       GAS (D/MCF)              1.49    1.77    1.86    1.94    1.98    1.99
195       CONDENSATE (D/B)         0.00    0.00    0.00   16.35   16.90   17.50
196       SULPHUR (D/LT)           0.00    0.00    0.00   27.50   27.50   30.00
                                  -----   -----   -----   -----   -----   -----
      REV.-NET BEFORE ROY.(MDLRS)
513       SALES GAS               268.2  1362.9  3852.1 10144.3 10030.7  9824.6
517       CONDENSATE                0.0     0.0     0.0    99.7   103.1   106.8
518       SULPHUR                   0.0     0.0     0.0   574.7   574.7   627.0
                                  -----   -----   -----   -----   -----   -----

519   GROSS REVENUES:             268.2  1362.9  3852.1 10818.7 10708.5 10558.4
520       INCR/DECR(PCT)            0.0   408.2   182.6   180.9    ¯1.0    ¯1.4
                                  -----   -----   -----   -----   -----   -----


      DATA BASE:  DEV
      DATE(DAY/MO./YR./HR./MIN.):    11  7   78   15   13
```

```
DIVISION:  DEVELOPMENT PROJECT EXAMPLE                                    ATTACHMENT B
*******************************************

                                CANADIAN OCCIDENTAL PETROLEUM LTD.
                                   CORPORATE FINANCIAL MODEL
                                  ROYALTIES AND OPERATING SUMMARY
                                *************************************

                                  1978      1979      1980      1981      1982      1983
        OIL AND GAS
        ============
       ROYALTIES (NET MDLRS):

529      SALES GAS (TOTAL):       80.8     420.6    1196.2    2967.7    2935.0    2871.1
530        1. CROWN               80.8     420.6    1196.2    2511.2    2468.4    2402.0
531        2. FRHLD + ORR          0.0       0.0       0.0     296.2     302.3     303.8
532        3. MINERAL TAX          0.0       0.0       0.0     160.3     164.3     165.3


537      CONDENSATE (TOTAL):       0.0       0.0       0.0      25.2      26.0      26.9
538        1. CROWN                0.0       0.0       0.0      11.7      12.1      12.5
539        2. FRHLD + ORR          0.0       0.0       0.0       8.7       9.0       9.3
540        3. MINERAL TAX          0.0       0.0       0.0       4.7       4.9       5.1

541      SULPHUR (TOTAL):          0.0       0.0       0.0     106.2     106.4     116.1
542        1. CROWN                0.0       0.0       0.0      28.7      28.7      31.3
543        2. FRHLD + ORR          0.0       0.0       0.0      50.3      50.3      54.9
544        3. MINERAL TAX          0.0       0.0       0.0      27.2      27.3      29.8

545      TOTAL ROYALTIES:         80.8     420.6    1196.2    3099.1    3067.4    3014.1
                                 -----     -----     -----     -----     -----     -----

548      NET REVENUE:            187.4     942.3    2655.9    7719.6    7641.1    7544.2
                                 -----     -----     -----     -----     -----     -----

       OPERATING COSTS:
 32            COST OF GOODS     330.5     403.5     524.4     716.8     777.3     808.9
521       TOT. OPERATING COST    330.5     403.5     524.4     716.8     777.3     808.9
                                 -----     -----     -----     -----     -----     -----

546    OPERATING INCOME:        ¯143.1     538.8    2131.5    7002.8    6863.8    6735.3
547    INCR/DECR(PCT.)             0.0    ¯476.7     295.6     228.5      ¯2.0      ¯1.9
                                 -----     -----     -----     -----     -----     -----

       DATA BASE:  DEV
       DATE(DAY/MO./YR./HR./MIN.):    11     7    78    15    13
```

```
DIVISION:  DEVELOPMENT PROJECT EXAMPLE                                   ATTACHMENT C
****************************************

                     CANADIAN OCCIDENTAL PETROLEUM LTD.
                         CORPORATE FINANCIAL MODEL
                             INCOME STATEMENT
                    ************************************

                                   1978     1979     1980     1981     1982     1983
     REVENUES:   (MDLRS)
519      GROSS REVENUES           268.2   1362.9   3852.1  10818.7  10708.5  10558.4
545      LESS:ROYALTIES            80.8    420.6   1196.2   3099.1   3067.4   3014.1
548      NET REVENUES             187.4    942.3   2655.9   7719.6   7641.1   7544.2
                                  -----    -----    -----    -----    -----    -----
603    TOTAL NET REVENUE          187.4    942.3   2655.9   7719.6   7641.1   7544.2
                                  -----    -----    -----    -----    -----    -----
     COSTS AND EXPENSES:
 32      COST OF GOOD SOLD        330.5    403.5    524.4    716.8    777.3    808.9
 37      CROWN LEASE RENTALS       39.6     39.6     39.6     39.6     39.6     39.6
612      D D + A                   32.7    118.2    391.8    779.0    859.0    837.7
                                  -----    -----    -----    -----    -----    -----
615    TOTAL NET COSTS            402.8    561.3    955.8   1535.4   1675.9   1686.2
                                  -----    -----    -----    -----    -----    -----
616    INCOME BEFORE TAX         ¯215.4    381.1   1700.1   6184.2   5965.3   5858.0
                                  -----    -----    -----    -----    -----    -----
     INCOME TAXES:
613      CURRENT FED. TAX        ¯389.6   ¯243.2    133.9   2167.7   2039.4   2202.2
614      CURRENT PROV. TAX       ¯109.4    ¯65.3     69.6    717.2    699.0    728.7
617      TOTAL CURRENT TAX       ¯499.0   ¯308.5    203.5   2884.9   2738.4   2930.9
618      DEFERRED                 371.7    522.3    727.1    319.5    320.3    123.4
                                  -----    -----    -----    -----    -----    -----
619    TOTAL TAXES               ¯127.3    213.8    930.6   3204.4   3058.8   3054.3
                                  -----    -----    -----    -----    -----    -----
620    NET INC. AFTER TAX         ¯88.1    167.3    769.4   2979.9   2906.5   2803.7
                                  -----    -----    -----    -----    -----    -----
621    CASH INC. FROM OPER.       316.4    807.7   1888.4   4078.3   4085.8   3764.8
                                  -----    -----    -----    -----    -----    -----

     DATA BASE:  DEV
     DATE(DAY/MO./YR./HR./MIN.):     11      7     78     15     13
```

```
DIVISION:  DEVELOPMENT PROJECT EXAMPLE                          ATTACHMENT D
****************************************

                          CANADIAN OCCIDENTAL PETROLEUM LTD.
                              CORPORATE FINANCIAL MODEL
                                 CASH FLOW STATEMENT
                          ***********************************

                                 1978      1979      1980      1981      1982      1983

      INCOME    (MDLRS)
620     NET INC. AFTER TAX      ¯88.1     167.3     769.4    2979.9    2906.5    2803.7
612     D D + A                  32.7     118.2     391.8     779.0     859.0     837.7
618     DEFERRED TAXES          371.7     522.3     727.1     319.5     320.3     123.4
                                -----     -----     -----     -----     -----     -----
621     CASH INC FROM OPER      316.4     807.7    1888.4    4078.3    4085.8    3764.8
                                -----     -----     -----     -----     -----     -----
      EQUITY ADDITIONS
                                -----     -----     -----     -----     -----     -----
659    TOTAL INFLOWS            316.4     807.7    1888.4    4078.3    4085.8    3764.8
                                -----     -----     -----     -----     -----     -----
      CAPITAL EXPENDITURES
650     GROSS EXPL. EXP.          8.0       0.0       0.0       0.0       0.0       0.0
658     CAPITAL EXP.           2200.0    2250.0    3700.0     400.0    1800.0     100.0
662     TAX PAY. VS. PROV.      ¯76.1    ¯190.5    ¯322.3   ¯1928.7    1808.7    ¯135.9
      DIVIDENDS
                                -----     -----     -----     -----     -----     -----
668    TOTAL OUTFLOWS          2131.9    2059.5    3377.7   ¯1528.7    3608.7     ¯35.9
                                -----     -----     -----     -----     -----     -----
669    NET CASHFLOW           ¯1815.6   ¯1251.7   ¯1489.4    5607.1     477.1    3800.7
                                =====     =====     =====     =====     =====     =====


      DATA BASE:  DEV
      DATE(DAY/MO./YR./HR./MIN.):       11     7    78    15    13
```

```
DIVISION: DEVELOPMENT PROJECT EXAMPLE                          ATTACHMENT E
*****************************************


                      CANADIAN OCCIDENTAL PETROLEUM LTD.
                          CORPORATE FINANCIAL MODEL
                           ASSETS AND LIABILITIES
                      ************************************

                                 1978     1979     1980     1981     1982     1983

ASSETS   (MDLRS)
******

    CURRENT ASSETS
625   CASH                    ¯4795.5  ¯6047.3  ¯7536.6  ¯1929.5  ¯1452.4   2348.3

629 TOTAL CURRENT ASSETS      ¯4795.5  ¯6047.3  ¯7536.6  ¯1929.5  ¯1452.4   2348.3
                                -----    -----    -----    -----    -----    -----
    FIXED ASSETS
630   P P + E                  5082.3   7332.3  11032.3  11432.3  13232.3  13332.3
631   LESS: CUM. D D + A         32.7    150.9    542.7   1321.7   2180.7   3018.4
146   UNPROVEN ACREAGE          ¯12.0    ¯12.0    ¯12.0    ¯12.0    ¯12.0    ¯12.0
148   LESS: CUM. RESERVES        10.0     10.0     10.0     10.0     10.0     10.0

632 TOTAL FIXED ASSETS         5027.6   7159.4  10467.6  10088.6  11029.6  10291.9
                                -----    -----    -----    -----    -----    -----

                                -----    -----    -----    -----    -----    -----
635 TOTAL ASSETS                232.1   1112.1   2931.0   8159.0   9577.2  12640.2
                                =====    =====    =====    =====    =====    =====

LIABILITIES AND EQUITY
**********************

    CURRENT LIABILITIES
636   INCOME TAX PAYABLE       ¯499.0   ¯308.5     13.7   1942.5    133.8    269.7

639 TOTAL CURRENT LIAB.        ¯499.0   ¯308.5     13.7   1942.5    133.8    269.7
                                -----    -----    -----    -----    -----    -----
641 DEFERRED INCOME TAX        1477.4   1999.7   2726.8   3046.3   3366.6   3490.0
                                -----    -----    -----    -----    -----    -----
645 TOTAL LIABILITIES           978.4   1691.2   2740.6   4988.8   3500.5   3759.7
                                -----    -----    -----    -----    -----    -----
    SHAREHOLDERS' EQUITY
643   COMMON                   ¯746.3   ¯579.1    190.4   3170.2   6076.7   8880.5

637 TOTAL EQUITY               ¯746.3   ¯579.1    190.4   3170.2   6076.7   8880.5
                                -----    -----    -----    -----    -----    -----
644 TOT LIABILITY AND EQUITY    232.1   1112.1   2931.0   8159.0   9577.2  12640.2
                                =====    =====    =====    =====    =====    =====


    DATA BASE: DEV
    DATE(DAY/MO./YR./HR./MIN.):      11    7   78   15   43
```



USING APL TO HELP MAKE DECISIONS IN A HURRY

Steven Graham
Dome Petroleum Limited
Calgary, Alberta

The Problems Encountered

Investment decisions in the petroleum industry involve:

- large dollar investments up to $2.5 billion for tar sands plants
- evolving government royalty and taxation rules which have significant economic impact
- high risk due to geological uncertainty
- quick decisions as the best deals are only offered for a short time

Some investment opportunities analyzed in my work are:

Land sales - every Tuesday, the industry bids for Crown mineral rights. These bids often total $5 million or more on several parcels. The geological information is sometimes received only the day before to minimize the risk of industrial espionage. Sometimes partners may approach our company the day of the sale with a proposed joint bid.

Corporate acquisitions - the company is constantly evaluating investments in other firms. Occasionally a competitive offer is made for the acquisition candidate before an analysis has progressed far enough. This calls for a great deal of analytical work done in a hurry to determine if a counter offer should be presented.

Beaufort Sea exploration - wells cost more than $26 million each and one summer's drilling program involves $130 million in total. There are many complex deals involving drilling funds, partners, purchase and disposition of mineral rights and investment in equipment to increase the drilling capabilities. At $500,000 per day for one drill ship, time is money.

Government royalty and taxation changes - the provinces often change the royalty formulas, which necessitates a modification to the computer models. An example is a reduction in Alberta Crown royalty on low production gas wells which changes marginally uneconomic wells to attractive investments. The production engineers want the formula as soon as the new policy is announced in the legislature but the details have not yet been finalized. Once the government's rules are released and understood, it is a simple matter to modify the program used in analyzing these investments.

The Solution for Typical Problems

Dome requires a discounted cash flow analysis for the majority of investment decisions. GOFER (Gas and Oil Field Economic Reporting) is an economic model which generates a discounted cash flow and other indicators. The program was developed originally by Myrna Iverson, I.P. Sharp Associates, and Gary Owen of Dome. Dome has a modified version of GOFER with various corporate assumptions added into the program. For example, price forecasts, royalty assumptions and income tax factors are specific to Dome's needs.

The original attempt at an economic analysis of drilling prospects involved the use of rule-of-thumb factors by the economics group. This involved too much crystal-ball gazing and gut feel. After GOFER was developed, the economics group could do a far more rigorous analysis with less effort. An important side benefit is that the production engineers can now do their own economic analyses and consequently are becoming quite knowledgeable about the economics of their projects. Some of the engineers work up several development cases and can fine-tune their strategy.

GOFER requires information about a particular oil or gas project such as production rates, forecast prices, operating costs, investments, tax and royalty burdens to generate statements showing details of the cash flow for each year of the project's life. The program is very simple to use. To facilitate the data entry phase, users are provided with a data preparation sheet which lists all the input variables in the system. Descriptions, units, and the default values are provided for each variable. A typical case is illustrated in the attached exhibits which show the data preparation sheet and the output.

GOFER has several advantages in our applications over competitive time-share models.

Easy to learn - the input form has about 60 variables of which only 15 would be used in most simple applications. A user can confidently use the program after two or three hours of instruction.

Relatively simple program - as black boxes go, GOFER is quite transparent. This allows the innovative user to perform analyses for which the program was not specifically designed. It also permits easy verification of results if a user has some doubts about the accuracy of the output. The simplicity is achieved at the cost of not being able to handle some complex cases. Since these occur infrequently, it is not too onerous to perform a manual adjustment.

Simple to modify - either I.P. Sharp or Dome can make changes in the program to incorporate new regulations or different assumptions. It often takes longer to properly interpret the new regulations than to modify the program. Special purpose versions of GOFER have been developed for analyzing various complex deals Dome has with drilling funds.

Cost of running program - a GOFER output for one case costs about $5 while competitive time-share programs are over $10 per case.

The Solution for Unique Problems

A considerable amount of analytical effort is spent in looking at unique projects unlike anything else the company has done previously. Some examples over the past year include participation in the Lorneterm LNG facility ($600 million), Shell tar sands consortium ($2.4 billion) and a Class 10 icebreaker ($150 million). Single purpose APL programs are quickly fabricated and run to facilitate evaluation of these projects.

On the tar sands project, Dome expressed interest at a relatively late stage of the negotiations and had to do its homework quickly to catch up. Since a number of cases had to be looked at, it was decided a computer program would be useful.

It was reported that Imperial Oil had spent six months developing their heavy oil model. It was rather ambitious to consider working up a tar sands model when the analysis was required in one week. About three man days were required of a programmer to write the model and another two man days of my time to debug and run the cases. Most of the difficulties arose from interpreting the government regulations on the royalty formula. Included in the Exhibits is a sample output from the program.

The Better Solution for Unique Problems

Dome has recently acquired a package designed for financial applications. It was developed by Tenneco of Houston and uses Honeywell APL.

It has powerful formatting functions to facilitate outputting tabular data. It also has functions with English names to allow users with no APL experience to develop and run programs.

A very useful feature of the package is the ability to do Fortran processing on recursive loops where APL is relatively weak. The example in the attached Exhibit shows this use where each year's opening investment pool depends on the prior year's depletion charge. This short example could be effectively handled with APL but a 20 year project involving tax loss carryforwards definitely favours Fortran handling of the loops.

Future Trends

Within Dome, more people actually involved in engineering, economic analysis and financial planning are being exposed to APL and writing simple programs to solve their problems. This is an ongoing process which one would expect to spread as more people see the value of an APL programming capability.

In production engineering, the engineers are running more development cases and can consider more alternatives using GOFER. The chances of finding the most economic solution are thus increased.

In financial planning, some problems which were impossible to analyze because of their complexity or data requirement, are now being programmed. Previously a rough cut had to suffice but now a more rigorous analysis is possible which leads to a better decision.

For major investment decisions with complex tax implications, a programming capability permits more cases to be run with higher accuracy.

EXHIBIT 1 - GOFER INPUT SHEET

I.P. Sharp Associates Ltd.

GOFER DATA

BASIC DATA

		Default
TITLE (Title for reports)		
Y (Starting year)		Present Year
LIFE (Maximum project life)		25
FLAG (1 Yes) (0 No) (Cut off at economic limit)		1
YOD (Year of discovery)		Present Year

PRODUCTION

OILPROD MBBLS/Year (Gross yearly production)		0
GASPROD MMCF/Year (Gross yearly production)		0
PCOND BBLS/MMCF (Proportion of condensate)		0
PPROP BBLS/MMCF (Proportion of propane)		0
PBUT BBLS/MMCF (Proportion of butane)		0
PSULPH LT/MMCF (Proportion of sulphur)		0

PRICES

PROIL $/BBL (Price of oil)		0
PRGAS ¢/MCF (Price of gas)		0
BTU BTU/CF (Gas BTU content)		1000
PRCOND $/BBL (Condensate price)		0
PRPROP $/BBL (Propane price)		0
PRBUT $/BBL (Butane price)		0
PRSULPH $/LT (Sulphur price)		0

ROYALTIES

LR $0 \le LR \le 1$ (Leasehold royalty)		0
MT $0 \le MT \le 1$ (Mineral Tax)		0
ROYALTY (Name of royalty scheme)		
POLD $0 \le POLD \le 1$ (Proportion of Old)		0
PNEW $0 \le PNEW \le 1$ (Proportion of New)		0
PRPOST $/BBL (Posted price of oil)		Per Current Regulation
PRSELECT $/BBL (Select price of oil)		Per Current Regulation
ORM $0 \le ORM \le 1$ (Overriding royalty on major stream)		0
ORS $0 \le ORS \le 1$ (Overriding royalty on subordinate stream)		0
ACCR ¢/MCF (Crown royalty allowance)		0
AORR ¢/MCF (Overriding royalty allowance)		0

CAPITAL

		Default
NWD (Number of Wells drilled)		0
TANW $M/Well (Tangible investment/well)		0
DEVINTW $M/well (Development Intangibles/well)		0
DEVINT $M (Development Intangibles)		0
EXPLINT $M (Exploration Intangibles)		0
DEI $M (Other depletion earning intangibles)		0
NDEI $M (Other non-depletion earning intangibles)		0
SUR30 $M (Inv. in surface facilities - 30% CCA)		0
SUR10 $M (Inv. in surface facilities - 10% CCA)		0
PLT20 $M (Investment in plant - 20% CCA)		0
PLN6 $M (Inv. in pipeline - 6% CCA)		0
OTHCAPn $M (Other tangible inv. - nth category)		0
OTHCAPRn $0 \leq OTHCAPRn \leq 1$ (CCA rate for other tangible)		0
OTHNC $M (Other non-cash deductions)		0

EXPENSES

OPEXPU $/MCF (Op. expenses per unit - major stream)		0
OPEXPW $/Well/Month (Op. expenses per well)		0
OPEXLR $M/Year (Op. expenses for lease rentals)		0
OPEXPT $M/Year (Op. expenses for plants)		0
OPEXPL $M/Year (Op. expenses for pipelines)		0
NW (Number of wells on stream)		0

OTHER PARAMETERS

OINC $M (Other Income)		0
DISR $0 \leq DISR \leq 1$ (Discount Rate)		0
PAROP 2 values (Discounts operating profit)		.5 Present Year
PARCAP 2 values (Discounts capital)		1 Present Year
PARTAX 2 values (Discounts taxes)		.5 Present Year
TAX (Name of the tax scheme)		'CANADA-ALTA.'
PRBCD (Deemed Price of BC Gas)		0
WINT $0 \leq WINT \leq 1$ (Working interest)		1

APRIL, 1977

EXHIBIT 2 - EXAMPLE OF GOFER RUN

```
      PRINT 'CURRENT'
TITLE      ←'EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATION ONLY'
OILPROD    ←0 8 TIMES 15 DECLINE 8
GASPROD    ←0 0 0 10 TIMES 800 DECLINE 10
PROIL      ←5 INFLATE 11.50
PRGAS      ←6 INFLATE 150
ROYALTY    ←'DOMEROY'
POLD       ←0 0 0 THEN .5
PNEW       ←0 0 0 THEN .5
ACCR       ←20
DEVINT     ←6 INFLATE 0 0 400 THEN 0
EXPLINT    ←6 INFLATE 0 500 THEN 0
SUR30      ←6 INFLATE 0 50 200 400 THEN 0
OPEXPU     ←6 INFLATE .10
OPEXPW     ←6 INFLATE 800
OPEXPT     ←6 INFLATE 0 0 0 THEN 250
NW         ←0 THEN 1
DISR       ←0.15
TAX        ←'CANADA-ALTA'
```

EXHIBIT 2 [CONT'D] - EXAMPLE OF A GOFER RUN

```
NORMAL
PLEASE ENTER THE CURRENT SELLING PRICE OF OIL IN DOLLARS PER BARREL
11.50

DATE: 07/13/78

                    EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATION ONLY

      PRODUCTION                                             NET CASH          NET CASH
    ------------                      OPER'G  OPER'G   CAP'L   BEFORE   INCOME    AFTER
      GAS     OIL  REVENUE  ROYALTY  EXPENSE  PROFIT     INV      TAX      TAX      TAX
YEAR MMCF  MBBLS        $M       $M       $M      $M      $M       $M       $M       $M
---- ----- ------- -------- -------- -------- ------- ------- -------- -------- --------
1978

1979            15      181                 10     171     583    (412)    (276)    (136)
1980            15      190                 11     179     674    (495)    (105)    (390)
1981   800      15     1629      545      404     680     476      203      266     (62)
1982   800      15     1725      583      429     713              713      347      365
1983   800      15     1826      624      454     747              747      396      351

1984   800      15     1933      668      482     784              784      438      345
1985   800      15     2047      714      511     823              823      478      345
1986   800      15     2167      763      541     864              864      515      349
1987   800      14     2274      806      574     894              894      544      351
1988   800      13     2387      852      608     927              927      573      354

1989   800      12     2508      902      645     961              961      602      359
1990   800      11     2637      956      683     997              997      632      365
1991   720      10     2518      916      707     894              894      584      310
1992   648       9     2405      878      733     793              793      536      257
1993   583       8     2297      842      762     693              693      489      204

1994   525       8     2193      806      793     594              594      442      152
1995   472       7     2095      772      826     496              496      396      100
1996   425       7     2001      739      862     399              399      351       48
1997   383       6     1911      708      901     302              302      305      (4)
1998   344       6     1825      677      943     205              205      260     (56)

1999   310       5     1743      648      988     107              107      215    (108)
2000   279       5     1665      620     1036       9                9      171    (162)
---- ----- ------- -------- -------- -------- ------- ------- -------- -------- --------
TOT  12690     239    42155    15019    13904   13231    1734    11498     8160     3338
PVRT                   9218     3142     2495    3582    1156     2425     1615      810
PVAT                   5715     2796     1547    1373     562      810

DISCOUNT RATE  15 PERCENT

                         BEFORE          AFTER
                           TAX            TAX
                           $M             $M
                       ---------       -------
NPV AT 10 PC.              3,885          1,267
NPV AT 15 PC.              2,425            810
NPV AT 20 PC.              1,566            531
NPV AT 25 PC.              1,037            355
NPV AT 30 PC.                699            243

DCF RATE OF RETURN  ****** PC.
```

EXHIBIT 3 - EXAMPLE OF SPECIAL PROGRAM

```
      )LOAD SHELL100
/SHELL100 SAVED 04/27/78 1:23:59 PM
      NCFSOLVE
SAME, CHANGE, ADD ARE VALID RESPONSES
(MAY USE ESCGEN×(1978$))
'TITLE'?
⎕: 'EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATIVE PURPOSES ONLY'
GENERAL INFLATION FACTORS, PER CENT?
⎕: 31ρ6
DO YOU WANT TO CHANGE INVESTMENTS?  Y
SAME, CHANGE, ADD ARE VALID RESPONSES
(MAY USE ESCGEN × (1978$))
'TITLE'?
⎕: SAME
INVESTMENTS:  ECE (10 P.C. D.B.)?
⎕: (0 0 6 6 10 10 10),25ρ0
     CLASS 10?
⎕: (5 0 5 10 10 5 5 20 20 20),22ρ0
     CLASS 28?
⎕: (20 30 300 1000 1000 1000 500 200 200),23ρ0
     CLASS 29?
⎕: (0 0 35 50),28ρ0
     EXP?
⎕: (8ρ20),24ρ0
     CDE?
⎕: (0 0 0 020 30 40),26ρ0
WORKING CAPITAL?
⎕: (8ρ0),100,23ρ0
     FRACTION RESOURCE?
⎕: 1
NEED 32 NUMBERS, YOU ONLY GAVE 1.
TRY AGAIN
⎕: 32ρ.6
ANSWER YES IF TAX SHIELDS CAN BE USED BY OTHER INCOME: YES
VOLUME, MB/D?
⎕: (7ρ0),50
          (50 80 100),22ρ115
PRICE, $/BBL?
⎕: 12×1.06*⍳32
OPERATING COST?
⎕: (7ρ0),400×1.06*⍳25
MINE CLOSING COST?
⎕: 250
INVEST MUST HAVE BEEN RUN TO SET PARAMETERS FOR ROYALTY AND TAX CALC
DO YOU WANT INVESTMENT TABLE PRINTED?  YES
     ROYALTY TABLE?  YES
     TAX TABLES?  YES
```

EXHIBIT 3 (CONT'D) - EXAMPLE OF SPECIAL PROGRAM
EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATIVE PURPOSES ONLY

TAX CALCULATION - INVESTMENT BREAKDOWN

```
            - - - - - - INVESTMENT BY TAX CLASS - - - - - -      WORKING   GRAND
      ECE    CL 10      CL 28  CL 29    EXP    CDE      TOTAL    CAPITAL   TOTAL

1978             5.00      20.00          20.00            45.00              45.00
1979                       30.00          20.00            50.00              50.00

1980    6.00     5.00     300.00  35.00   20.00           366.00             366.00
1981    6.00    10.00   1,000.00  50.00   20.00  20.00  1,106.00           1,106.00
1982   10.00    10.00   1,000.00          20.00  30.00  1,070.00           1,070.00
1983   10.00     5.00   1,000.00          20.00  40.00  1,075.00           1,075.00
1984   10.00     5.00     500.00          20.00           535.00             535.00

1985            20.00     200.00          20.00           240.00             240.00
1986            20.00     200.00                          220.00    100.00   320.00
1987            20.00                                      20.00              20.00
1988
1989

1990
1991
1992
1993
1994

1995
1996
1997
1998
1999

2000
2001
2002
2003
2003

2005
2006
2007
2008
2009                                                              (100.00) (100.00)


PV 0   42.00   100.00   4,250.00  85.00  160.00 90.00  4,727.00           4,727.00
PV 8   29.22    61.21   2,938.96  67.07  119.44 62.69  3,278.59     43.13 3,321.72
PV10   26.84    54.80   2,696.99  63.40  111.91 57.54  3,011.48     39.51 3,050.99
PV12   24.71    49.29   2,480.61  59.99  105.15 52.91  2,772.66     35.35 2,808.01
PV15   21.91    42.39   2,197.08  55.34   96.24 46.80  2,459.77     29.26 2,489.02
PV20   18.10    33.66   1,812.91  48.60   84.07 38.45  2,035.79     20.91 2,056.70

SHELL100 INVEST
BUSDEV  1978 07 13
```

EXHIBIT 3 (CONT'D) - EXAMPLE OF SPECIAL PROGRAM
EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATIVE PURPOSES ONLY

ROYALTY - ALBERTA COAL FORMULA

	ADDED INVEST (1)	*INT ALLOW* (2)	*AMORT BASE* (3)	*AMORT OF CAP* (4)	*OPER ALLOW* (5)	*ANN COST ALLOW* (6)	*WORK CAP ALLOW* (7)	*ANN INVEST ALLOW* (8)	*PROJECT FACTOR* (9)	*ROYALTY RATE* (10)	*REVENUE* (11)	*ROYALTY* (12)
1978	45.0	2.0	47.0					47.0				
1979	50.0	6.5	103.5					103.5				
1980	366.0	24.2	458.7					458.7				
1981	1,106.0	88.8	1,603.5					1,603.5				
1982	1,070.0	192.5	2,866.0					2,866.0				
1983	1,075.0	306.3	4,247.3					4,247.3				
1984	535.0	406.3	5,188.6					5,188.6				
1985	240.0		5,747.2	287.4	466.2	753.6	113.0	5,860.2	0.129	0.043	349.1	15.1
1986	220.0		6,014.0	300.7	492.9	793.6	119.0	6,133.0	0.181	0.025	592.0	14.8
1987	20.0		6,076.7	303.8	521.2	825.1	123.8	6,200.4	0.216	0.038	784.4	29.4
1988			6,119.2	306.0	551.2	857.2	128.6	6,247.8	0.242	0.050	956.2	47.8
1989			6,162.1	308.1	583.1	891.2	133.7	6,295.7	0.252	0.050	1,013.5	50.7
1990			6,205.2	310.3	616.8	927.0	139.1	6,344.2	0.262	0.050	1,074.4	53.7
1991			6,248.6	312.4	652.5	965.0	144.7	6,393.4	0.272	0.050	1,138.8	56.9
1992			6,292.4	314.6	690.4	1,005.0	150.8	6,443.1	0.283	0.050	1,207.1	60.4
1993			6,336.4	316.8	730.6	1,047.4	157.1	6,493.5	0.293	0.050	1,279.6	64.0
1994			6,380.8	319.0	773.2	1,092.2	163.8	6,544.6	0.304	0.050	1,356.3	67.8
1995			6,425.4	321.3	818.3	1,139.6	170.9	6,596.4	0.316	0.050	1,437.7	71.9
1996			6,470.4	323.5	866.1	1,189.6	178.4	6,648.9	0.327	0.050	1,524.0	76.2
1997			6,515.7	325.8	916.8	1,242.6	186.4	6,702.1	0.339	0.050	1,615.4	80.8
1998			6,561.3	328.1	970.6	1,298.6	194.8	6,756.1	0.351	0.050	1,712.4	85.6
1999			6,607.2	330.4	1,027.6	1,357.9	203.7	6,810.9	0.363	0.050	1,815.1	90.8
2000			6,653.5	332.7	1,087.9	1,420.6	213.1	6,866.6	0.375	0.050	1,924.0	96.2
2001			6,700.1	335.0	1,152.0	1,487.0	223.0	6,923.1	0.388	0.050	2,039.4	102.0
2002			6,747.0	337.3	1,219.8	1,557.2	233.6	6,980.5	0.401	0.050	2,161.8	108.1
2003			6,794.2	339.7	1,291.8	1,631.5	244.7	7,038.9	0.413	0.050	2,291.5	114.6
2004			6,841.8	342.1	1,368.0	1,710.1	256.5	7,098.3	0.426	0.050	2,429.0	121.5
2005			6,889.6	344.5	1,448.8	1,793.3	269.0	7,158.6	0.440	0.050	2,574.8	128.7
2006			6,937.9	346.9	1,534.5	1,881.4	282.2	7,220.1	0.453	0.050	2,729.2	136.5
2007			6,986.4	349.3	1,625.3	1,974.6	296.2	7,282.6	0.466	0.050	2,893.0	144.6
2008			7,035.3	351.8	1,721.6	2,073.3	311.0	7,346.3	0.480	0.050	3,066.6	154.3
2009			7,084.6	354.2	1,823.6	2,177.8	326.7	7,411.3	0.493	0.054	3,250.6	174.6
PV 0	4,727.0	1,026.6	177,347.6	8,141.6	24,950.7	33,092.3	4,963.8	182,311.5			43,216.0	2,146.9
PV 8	3,278.6	678.7	50,671.3	2,048.8	5,174.9	7,223.7	1,083.6	51,754.9			8,640.6	419.5
*PV*10	3,011.5	615.6	39,557.9	1,537.0	3,720.0	5,256.9	788.5	40,346.4			6,138.3	295.9
*PV*12	2,772.7	559.7	31,534.5	1,174.8	2,734.2	3,909.0	586.4	32,120.9			4,455.4	213.2
*PV*15	2,459.8	487.1	23,214.3	809.5	1,789.0	2,598.5	389.8	23,604.0			2,858.1	135.1
*PV*20	2,035.8	390.4	14,972.7	465.4	960.2	1,425.6	213.8	15,186.6			1,482.8	68.7

*SHELL*100 *ROYCOAL*
BUSDEV 1978 07 13

```
                EXHIBIT 3 (CONT'D) - EXAMPLE OF SPECIAL PROGRAM
             EXAMPLE RUN - NUMBERS ARE FOR ILLUSTRATIVE PURPOSES ONLY

                                        NET CASH FLOW  (MM$)

             SALES REVENUE                OPER               OPER     INVEST  TAX NOT DEFERRED     DOME POSITION      PROJECT BASIS
         MB/D    $/BBL       MM$       COST  ROYALTY       PROF      + W/C       TAX        NCF        TAX       NCF      TAX        NCF

1978             12.72                                                45.0    (14.4)     (30.6)               (45.0)                (45.0)
1979             13.48                                                50.0    (17.7)     (32.3)               (50.0)                (50.0)

1980             14.29                                               366.0   (101.3)    (264.7)              (366.0)               (366.0)
1981             15.15                                             1,106.0   (331.8)    (774.2)    (465.2)   (640.8)             (1,106.0)
1982             16.06                                             1,070.0   (402.4)    (667.6)    (402.4)   (667.6)             (1,070.0)
1983             17.02                                             1,075.0   (447.2)    (627.8)    (447.2)   (627.8)             (1,075.0)
1984             18.04                                               535.0   (338.0)    (197.0)    (338.0)   (197.0)               (535.0)

1985     50.0    19.13     349.1      424.0     15.1     (90.1)      240 0   (274.5)     (55.5)    (274.5)    (55.5)               (330.1)
1986     80.0    20.27     592.0      449.4     14.8      127.8      320.0   (146.6)       54.3    (146.6)      54.3     21.0      (213.2)
1987   100.00    21.49     784.4      476.4     29.4      278.6       20.0     (4.8)      263.4      (4.8)     263.4     45.7        212.9
1988    115.0    22.78     956.2      505.0     47.8      403.4                 59.2      344.2       59.2     344.2     66.2        337.2
1989    115.0    24.15   1,013.5      535.3     50.7      427.6                 63.4      364.2       63.4     364.2     70.1        357.5

1990    115.0    25.60   1,074.4      567.4     53.7      453.2                151.0      302.2      151.0     302.2     74.3        378.9
1991    115.0    27.13   1,138.8      601.5     56.9      480.4                201.1      279.3      201.1     279.3     78.8        401.6
1992    115.0    28.76   1,207.1      637.5     60.4      509.2                214.0      295.3      214.0     295.3     83.5        425.7
1993    115.0    30.48   1,279.6      675.8     64.0      539.8                227.4      312.4      227.4     312.4     88.5        451.3
1994    115.0    32.31   1,356.3      716.3     67.8      572.2                241.5      330.7      241.5     330.7     93.8        478.4

1995    115.0    34.25   1,437.7      759.3     71.9      606.5                256.3      350.2      256.3     350.2     99.5        507.1
1996    115.0    36.31   1,524.0      804.9     76.2      642.9                271.9      371.0      271.9     371.0    105.4        537.5
1997    115.0    38.49   1,615.4      853.2     80.8      681.5                288.4      393.1      288.4     393.1    111.8        569.7
1998    115.0    40.79   1,712.4      904.4     85.6      722.4                305.8      416.6      305.8     416.6    118.5        603.9
1999    115.0    43.24   1,815.1      958.6     90.8      765.7                324.3      441.4      324.3     441.4    125.6        640.1

2000    115.0    45.84   1,924.0    1,016.1     96.2      811.7                343.8      467.8      343.8     467.8    133.1        678.6
2001    115.0    48.59   2,039.4    1,077.1    102.0      860.4                364.5      495.8      364.5     495.8    191.3        669.1
2002    115.0    51.50   2,161.8    1,141.7    108.1      912.0                386.4      525.5      386.4     525.5    328.8        583.2
2003    115.0    54.59   2,291.5    1,210.2    114.6      966.7                409.7      557.0      409.7     557.0    348.5        618.2
2004    115.0    57.87   2,429.0    1,282.9    121.5    1,024.7                434.3      590.4      434.3     590.4    369.5        655.3

2005    115.0    61.34   2,574.8    1,359.8    128.7    1,086.2                460.4      625.8      460.4     625.8    420.5        665.7
2006    115.0    65.02   2,729.2    1,441.4    136.5    1,151.4                488.0      663.3      488.0     663.3    488.0        663.3
2007    115.0    68.92   2,893.0    1,527.9    144.6    1,220.4                517.3      703.1      517.3     703.1    517.3        703.1
2008    115.0    73.06   3,066.6    1,619.6    154.3    1,292.7                548.3      744.4      548.3     744.4    548.3        744.4
2009    115.0    77.44   3,250.6    1,716.7    174.6    1,359.3   (100.00)     579.9      779.4      579.9     779.4    579.9        879.4

PV 0 2,760.0            43,216.0   23,262.6  2,146.9   17,806.6    4,727.0   5,058.2    8,021.4    5,058.2   8,021.4  5,107.8      7,971.8
PV 8    682.5            8,640.6    4,799.0    419.5    3,422.1    3,321.7   (111.5)      255.0    (100.2)     243.7    816.9      (716.5)
PV10    508.4            6,138.3    3,445.1    295.9    2,397.3    3,051.0   (388.7)    (225.4)    (375.4)   (238.8)    548.5    (1,202.1)
PV12    385.5            4,455.4    2,528.9    213.2    1,713.4    2,808.0   (542.0)    (517.3)    (526.9)   (532.4)    377.0    (1,471.6)
PV15    262.3            2,858.1    1,651.7    135.1    1,071.3    2,489.0   (640.9)    (747.6)    (623.5)   (765.0)    223.9    (1,641.7)
PV20    147.3            1,482.8      884.2     68.7      529.9    2,056.7   (646.3)    (859.6)    (626.0)   (879.9)    104.0    (1,630.8)

INTERNAL RATE OF RETURN

SHELL100  NCF
BUSDEV  1978 07 13
```

EXHIBIT 4 - EXAMPLE OF BETTER METHOD

FORECAST OF RESERVES,PRODUCTION AND DISCOVERIES
SMALL PETROLEUM COMPANY
1978---1982

	CONV/OIL MBBLS	FRONT/OIL MBBLS	GAS LIQ MBBLS	SULPHUR MBBLS	CONV/GAS BCF	FRONT/GAS BCF	SUBTOTAL MBBLS	OTHERGAS BCF	TOTAL MBBLS
JAN1/78 RESERVES	100,000		10,000	800	2,000	1,000	485,800	50	492,050
78 PRODUCTION	5,000		300	3	100		17,803	3	18,178
78 ADDITIONS	10,000	4,000	600		200	1,100	177,100		177,100
JAN1/79 RESERVES	105,000	4,000	10,300	797	2,100	2,100	645,097	47	650,972
79 PRODUCTION	5,250		350	3	105		18,728	3	19,103
79 ADDITIONS	10,500	6,000	900		210	1,210	194,900		194,900
JAN1/80 RESERVES	110,250	10,000	10,850	794	2,205	3,310	821,269	44	826,769
80 PRODUCTION	5,512		400	3	110		19,665	3	20,040
80 ADDITIONS	10,624	8,000	1,000		220	1,331	213,499		213,499
JAN1/81 RESERVES	115,362	18,000	11,450	791	2,315	4,641	1,015,103	41	1,020,228
81 PRODUCTION	5,768		450	3	116		20,721	3	21,096
81 ADDITIONS	11,536	10,000	1,300		232	1,464	234,836		234,836
JAN1/82 RESERVES	121,130	28,000	12,300	788	2,431	6,105	1,229,218	38	1,233,968
82 PRODUCTION	6,056		480	3	122		21,789	3	22,164
82 ADDITIONS	12,113	12,000	1,500		243	1,611	257,363		257,363

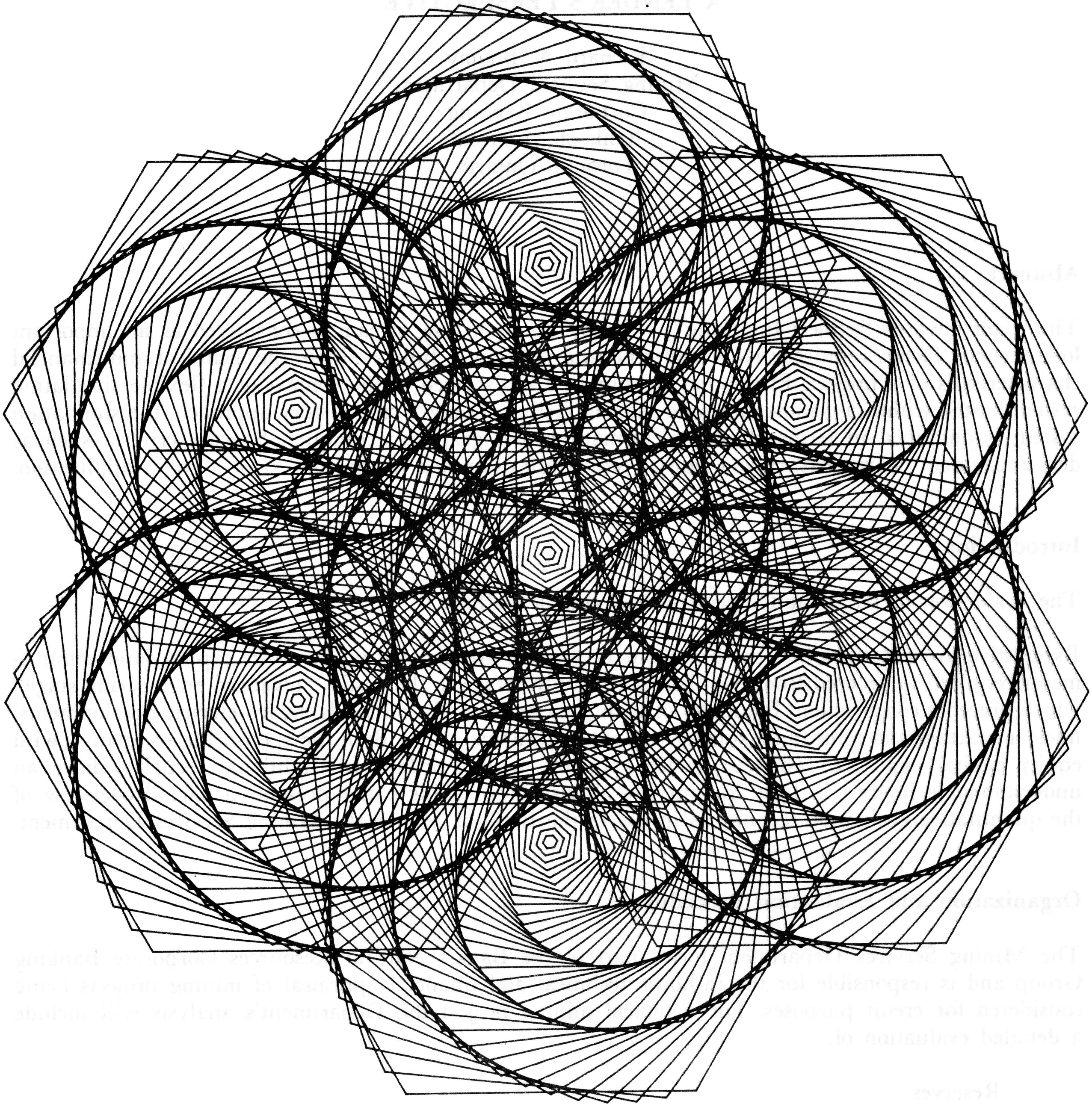
CONVERSION RATE FOR GAS IS 8 MCF PER BARREL

CONVERSION RATE FOR SULPHUR IS 3 BBLS PER TON

JAN1/78 RESERVES FROM GAS UTILIZATION AND OIL RESERVOIR ENGINEERING

CALCULATION OF DEPLETION
SMALL PETROLEUM LIMITED
$000

JAN1/78 INTANGIBLE ASSETS	100,000
78 DEPLETION	3,665
	=======
78 ADDITIONS	20,000
JAN1/79 INTANGIBLE ASSETS	116,335
79 DEPLETION	3,377
	=======
79 ADDITIONS	25,000
JAN1/80 INTANGIBLE ASSETS	137,958
80 DEPLETION	3,296
	=======
80 ADDITIONS	30,000
JAN1/81 INTANGIBLE ASSETS	164,662
81 DEPLETION	3,361
	=======
81 ADDITIONS	35,000
JAN1/82 INTANGIBLE ASSETS	196,301
82 DEPLETION	3,480
	=======
82 ADDITIONS	40,000



TECHNICAL AND FINANCIAL EVALUATION OF CAPITAL PROJECTS A LENDER'S PERSPECTIVE

Richard A. Boulay
Mining Services Department
Corporate Banking
Bank of Montreal
Toronto, Ontario

Abstract

The trend towards "project financing", whereby the cash flow stream is considered as an important loan security factor, requires that a lender be in a position to fully evaluate the technical and financial characteristics of a project. Evaluation techniques are discussed in the context of a flexible simulation system hosted within a commercial time-sharing facility. Using a case history approach, the simulation system is described in terms of coding, formatting and graphic techniques. Specific criteria such as debt repayment characteristics and rate of return are examined in the context of a credit application.

Introduction

The evaluation of large capital projects is a complex affair.

Exhaustive investigations into the behaviour of projects are dictated by the large capital fund commitments required to proceed from the "feasibility study" stage to the costly "advanced engineering stage" which precedes construction. Examples of such projects abound - oil sands, hydro-electric developments, nuclear reactors and mine/mill/smelter complexes to name a few. Both the sponsoring company (with equity capital exposure) and the lender (possibly with shortfalls in traditional security) have an understandable interest in fully quantifying all risk areas. This paper presents a partial overview of the quantitative evaluation techniques in use in the Bank of Montreal's Mining Services Department.

Organization and Evaluation Procedure

The Mining Services Department forms part of the Bank's Natural Resources Corporate Banking Group and is responsible for the in-house technical and financial appraisal of mining projects being considered for credit purposes. For a typical mining project the Department's analysis will include a detailed evaluation of:

Reserves
Ore grades
Mining methods
Construction plans
Processing techniques
Transportation factors
Environmental and other compliance aspects
Marketing arrangements
All capital and operating cost considerations
Management expertise

This evaluation is conducted in close cooperation with the mining company's technical and financial staff and with other departments within the Bank.

Although each mining project is unique it may be generally observed that the evaluation process falls into two stages - a technical evaluation and a stage which might be loosely described as consisting of the transformation of technical information into financial results. Our department performs both the technical analysis and the subsequent reduction of analytical results into a cash flow and debt servicing summary for eventual consideration by qualified credit managers within the Natural Resources Corporate Banking Group.

The transformation of technical information into financial results in a form suitable for credit appraisal can be both complicated and tedious, especially since it is always desirable to perform numerous analyses of the cause and effect relationships between technical factors and debt repayment characteristics or other measures of project performance. Since, in practice, at least 10 and sometimes as many as 150 input parameter scenarios are desirable, the evaluation process begs a data processing solution.

EDP Decision

Approximately 2 years ago a decision was made to develop a flexible simulation system which would accommodate the complexities of integrated mining projects and, at the same time be reasonably priced in proportion to the costs usually allocated for the in-house appraisal of such projects.

The writer was charged with investigating and implementing such a "system" within a time period of one year. Specific decision criteria used in rating and selecting a computing option will be described later. At this point it suffices to say that the two key implementation criteria were:

- Interactive capability
- APL programming language

Having selected these two major characteristics, the implementation of the evaluation system consisted initially in designing broad specifications for a simulation package. This procedure was aborted when most of the design criteria were found to be resident in the AIDS facility supplied by I.P. Sharp Associates. The prime considerations in electing to use the AIDS simulation facility were:

- A partly conversational, automatic and self-documenting filing system for case simulations.
- a flexible, simple formatting capability.
- total recourse to primitive APL.

A fully operational project evaluation status was achieved within 6 months. The AIDS facility is still being used although a customized simulation system is being developed and is expected to be operational by year-end 1978.

A Narrative Description of the Evaluation Process

Before proceeding to detail case history descriptions it would be useful to first consider the general structure of the output results of a typical evaluation. This is best described in terms of extensions to a standardized cash flow statement.

At the top of such statements we usually find a row which details sales revenues, or metal revenues in the case of mining projects. Near the bottom of a typical cash flow statement, we would expect to find, following the allocation of funds to repay debt financing, a net cash flow summary and possibly measurements of project performance such as rate of return or the present value of cash flow streams at arbitrary discount rates. Graphically speaking, our evaluation procedure simply extends the cash flow statement upwards to encompass the myriad technical factors which apply to mining operations. A typical first line would represent the number of tons mined on an annual basis. Subsequent lines would describe reserve grades, recovery rates, metal prices, and other factors which eventually lead to revenue figures.

Below the derived revenue line the calculations are primarily concerned with cost and tax matters. Progressing downward in the statement, we generate net available profits which may be of use in the case of, say, an income debenture or term preferred based loan or alternately the evaluation may simply require the derivation of a simple cash flow figure. Subtracted from the cash flows are capital replacement expenditures required to sustain the operation through its planned life.

Subsequent to these deductions a decision is made as to whether or not the evaluation will consider a proportion of cash flow remaining as being dedicated to debt or whether a fixed repayment schedule will be imposed.

Following debt servicing requirements, the excess cash flow is totalled, cumulated and usually subjected to rate of return procedures. At this point it is interesting to note that the decision to evaluate the project by means of debt repayment using a proportion of cash flow as opposed to a fixed payment scheme will result in considerably different coding techniques. A decision to proceed along the former route will necessitate the structuring of one major loop within the evaluation function and consequently will require the tedious use of indices. A decision to impose a fixed repayment schedule will usually obviate the need for loops since interest payments can be pre-calculated throughout the life of the project without recourse to year end "debt outstanding" balances.

Following the evaluation, the results are submitted formally for a credit decision. This submission usually contains an extensive summary of numerous case simulations. In short, the credit decision is made with the benefit of an exhaustive quantitative exposition of all factors to which the project is sensitive.

Building A Simple Capital Project Estimating Model

I will now describe, in a general way, the sequential construction of a data file (representing a simple, generalized capital model), the manipulation of the file according to a user-defined function and the subsequent formatting of the file in a manner suitable for inclusion in a report. Due to space constraints this discussion will not detail file management or formatting procedures.

The label block in the upper left-hand corner of Table 1 refers to CASE 0 (the BASE CASE) of file *TERM* and that, furthermore, this case is defined over years 1 through 10.

The row labels down the left hand side of Table 1 are entered using a conversational function. For clarity those lines which constitute input variables are marked with a quad (⎕).

For ease of reference, the row labels are numbered sequentially, in this case from 1 through 27. Later we shall see that the rows in our data matrix can be referenced by using the assigned row numbers.

The numeric data matrix in this case is defined as a 27 by 10 matrix filled with zeros. A conversational function permits the allocation of the desired decimal accuracy on a row-by-row basis.

Table 2 illustrates our data matrix following the specification of input variables.

Note that output variables are still represented by zeros. These will be replaced with new values using the input variables according to logic contained in the user-defined function.

Before examining this function I would first draw attention to the row name block at the left hand side of Table 2. Lines 1 through 5 are input variables. Line 1 is simply a year label and it is useful when printing out unformatted copies of the Base Case. Line 2 is the interest rate which we have arbitrarily assumed to be 10%. Line 3 represents the total amount of funds required to initiate the project. This can be considered as representing \$10,000, \$100,000 \$10,000,000 or \$100,000,000. The value of \$10,000 is assumed to represent total funds required to permit the operation to begin on day 1 of year 1. This includes construction costs, engineering costs, capitalized interest and working capital. Line 4 is the assumed debt to equity ratio and is 75%, i.e., 75% debt and 25% equity. Line 5 is the operating profit which is arbitrarily assumed to be \$2,250. As an aside, I might mention that this sum was selected arbitrarily to yield a 15% rate of return on equity. Line 22 is another input variable and it defines the percentage of cash flow generated by the project to be subsequently used to reduce the outstanding debt.

The function *∇CAPMODEL* (Figure 1) defines the logical relationships between the 27 lines in our data matrix. It utilizes 5 other functions, primarily for purposes of reading input variables and writing derived variables into the data matrix for eventual assignment to files.

Preliminary to dissecting *∇CAPMODEL* (with reference to Tables 1, 2 and eventually Table 3) it is worth noting that this function, as are most of our evaluation functions, replete with coding inefficiencies. There are several reasons for this:

i) the author is not a professional programmer,

ii) the function was written under severe time constraints, and

iii) the function is not a "production program" in the accepted sense since it was only used approximately 20 times before being relegated to storage.

In line 1 of the function *∇CAPMODEL* the secondary function *∇READID* reads in our six input variables and renames them with the prefix A, i.e., *A*5. For example, the variable *A*5 is now a vector containing the 10 elements in row 5 of the data matrix.

Line 2 simply sets some counters. The variable *CCAPOOL* will be used later in tax calculations while the counter *I* will be used in indexing a loop.

The statement in line 3 serves a dual purpose. First to zero out all derived variables. This is useful if we forget to redefine a variable during a function modification by preventing old values which are no longer material to be carried forward into the data matrix and subsequently formatted into a report. The second purpose is to define some variables as zero-filled vectors. This is mandatory for a subsequent looping operation where we will be referring to indexed components of the defined variables. A failure to define zero filled vectors at this point will result in *VALUE ERRORS* within the loop.

Line 4 generates a depreciation vector over 10 years. The formula contained in line 4 is a general case since, rather than referring to a number 10 (representing straight line depreciation over 10 years), we refer to a value of ⍴*A*1.

In line 5 the variable *CCASTRING* defines the Capital Cost Allowance vector assuming a 30% rate. The coding in this line is only interesting insofar as it uses the power primitive with the index generator in the right-hand argument.

Line 6 is a calculation of the interest due in year 1, a preliminary to entering the major loop within this function. It should be noted that a rather simplistic interest calculation is used here, i.e., the interest rate is applied to the maximum outstanding balance during the whole year. In practice, more accurate, albeit more complicated, interest calculations are often used.

Line 7 calculates the interest during the second and subsequent years. The interest due is based on the amount outstanding at the end of the previous year. This line is skipped over when $I = 1$, i.e., during the first pass through.

Line *L2* (Line 8) defines pretax income, a rather simple arithmetic operation. It is only notable in that line *L2* is the primary loop starting point for the first pass.

Line 9 contains three statements, all referring to the Capital Cost Allowance calculations specific to Canadian situations. The scalar variable *CCAPOOL* is defined by simple addition of an indexed value of the variable *CCASTRING* to the storage variable *CCAPOOL*. The second statement in line 9 is the actual assignment of the value of the Capital Cost Allowance and is interesting from a coding viewpoint in that the minimum primitive ⌊ is used to select the lesser of the variable *CCAPOOL* or another defined arithmetic value. It is further selected to return the greater of that value or zero, the result being stored in the variable *A9* at the correct index position. The third statement simply adjusts the scalar variable *CCAPOOL* according to the amount used in the second statement. It should be noted here that all three statements can be written as one expression, albeit at the expense of clarity when reviewing the function after several months.

The three statements on line 10 are tax related calculations and are not particularly interesting as to either mathematics or coding.

Line 11 calculates deferred tax paid and is interesting from a coding viewpoint; it uses both a minimum ⌊ and maximum ⌈ selector and, in addition, makes efficient use of the plus reduction procedure in combination with the index generator operating on the value of *I*. This is a powerful statement which obviates the need for several scalar storage variables and at least one nested loop.

The statements in lines 12, 13 and 14 are self-explanatory.

On line 15 the value of the net cash flow is defined.

The test on line 16 checks to make sure that all of the columns in the data matrix have been subjected to the looping operation and then the sequence is passed to line 17 which contains two simple arithmetic operations.

On line 18 the function *∇GET* simply gets the function *∇ROR* from the files. Function *∇ROR* is a rate of return function which is applied in lines 19 and 20 to determine the rates of return on equity and on the total funds employed respectively.

In line 21 we determine the present value of the net cash flow at an arbitrarily selected discount rate of 12%. We also calculate the government tax take at an arbitrarily selected discount rate of 8% as indicated in line 22.

In line 23 the variable *LIST* is defined and will provide one argument to the function ∇*STORE*. The function ∇*STORE* is employed in line 24 to define a matrix of derived variables comprising the catenated vectors of the right argument. The left hand argument (*LIST*) will serve as an address list for storing the values of the derived variables in the data matrix.

In line 25 the function ∇*WRITE* takes the updated data matrix and sends it out to file.

Table 3 is the result of subjecting the data matrix to the defined function ∇*CAPMODEL*. Note that all the output variable positions have been filled according to the rules outlined in the function.

Table 4 represents the data matrix formatted and decorated in a style suitable for inclusion in a report.

Selective use of graphics is made to clarify the behaviour of derived results to variations in input values.

Figures 2 and 3 illustrate the effective use of graphics. They show the effect of variance in operating profit on the project rate of return on equity for both a conventional term loan (∇) and an income debenture (⎕) as defined in Canadian tax law. The term loan results were generated using the function ∇*CAPMODEL* while the income debenture results were acquired using a modification of ∇*CAPMODEL*.

Figure 2 assumes normal tax conditions and shows that from an equity *ROR* viewpoint there is no material advantage to be gained by the selection of income debenture financing over a term loan or vice-versa.

Figure 3 assumes that the sponsor company carries a $6MM tax loss into the project. At very low operating efficiencies (where debt is usually not repaid in full) the income debenture financing is shown to offer some advantage, however, as project return climbs into acceptable ranges the differences in equity *ROR* become negligible.

U.S. Coal Project - A Case History

Table 5 represents a case history of an Eastern U.S. coal acquisition loan evaluation. Only cosmetic changes have been made - client identifiers have been removed and numeric changes have been made to certain input variables. Note that the line numbering down the left side of the table is not sequential. This is typical of most appraisals and simply indicates that numerous changes have been made during the evaluation process.

Lines 101 through 110 allow for the mining and processing of variable tonnages of coal; partly on a company-mined basis as indicated in line 101 at specified unit costs as indicated in line 102. Lines 104 and 105 provide for the mining of coal on a contract basis.

Lines 103 and 106 reflect the coal yield, the percentage of marketable product following processing.

Line 107 is a derived variable and indicates the number of tons available for sale. This saleable tonnage may or may not include coal that has been processed through a washing plant. The data file also provides for consideration of tonnages washed at specified costs as indicated in lines 109 and 110.

Line 4 is an input variable specifying the coal price. This line is, in effect, the beginning of the project income statement.

In line 6, tonnage and price is translated into a gross sales figure and is subject to various cost deductions as outlined in lines 7 through 21. These include operating costs as derived from the "per ton" figures, sales, general and administrative costs, royalties, overriding royalties (where applicable), depreciation, State taxes, interest expenses, depletion allowance and taxes. These deductions give rise to a net income and, after adding in non-cash deductions, a project cash flow as indicated in line 23. Subtracted from the cash flow figures are working capital requirements, anticipated plant and equipment expenditures, non-capital equipment replacements and various other expenses which are not eligible for tax credits.

Lines 54, 112 and 27 indicate that this data file and the functions used to manipulate it will evaluate three loans. Line 54 accepts input denoting the loan requested for purposes of purchasing the project assets. The loans detailed in line 112 are for purposes of expanding the operation following asset acquisition by the client. Line 27 is also an input line and represents forecasted working capital loan requirements.

The debt interest on the amounts outstanding is calculated by a user-defined function and is expressed in lines 15 and 16. Line 15 incorporates the interest expense arising from the purchase loan and the working capital injection loan, whereas the interest in line 16 is allocated to the capital equipment loan.

Our previous example stipulated that the loan would be repaid using a certain proportion of cash flow generated by the project. In this case two loan schedules are specified. These are totalled in line 30 and are compared with the amounts available for payment in line 31. Line 32 represents the net cash flow to the project after all expenses, interest charges, capital replacement expenses and loan payments. This sum is cumulated in line 33, present valued at an arbitrary discount rate of 12% in line 62, and reduced to a rate of return on total funds employed amount in line 63.

This is an interesting case history since it describes not only a project evaluation example for bank credit purposes but also gives some indication of how the bank was able to assist its client in determining a reasonable purchase price for the coal property.

The Cost of Operating a Simulation System

Table 6 outlines the order-of-magnitude computing costs incurred in evaluating six mining projects. The six projects outlined in the table vary in capital costs from a minimum of $40MM to $350MM. A glance at the "Computer Costs" column shows that these costs are not necessarily related to the amount of the loan being analyzed. Rather, the costs are a function of project complexity and the number of case simulations required for an adequate assessment of each project's technical and financial characteristics. Although the value of these computer costs as a proportion of total in-house evaluation costs on an annual basis cannot be disclosed, it is generally accurate to state that computer-related costs vary between 1% and 5% of total direct project evaluation costs.

Obviously the benefit/cost ratio for an effective interactive project evaluation capability is very high even when intangibles such as competitive marketing advantages are ignored.

Reflections on the Implementation and Use of a Computerized Simulation System

It has been two years since a decision was taken to acquire an interactive project evaluation capability. In retrospect, several interesting observations on the implementation of this system are made.

It was assumed that approximately 1 year would be required to develop an efficient analytical capability with sufficient flexibility to accommodate the great variety of mining projects presented to the Bank's Mining Services Department. In fact, this capability was attained within approximately 6 months. A secondary objective, namely to develop a number of customized functions to aid in formatting and filing operations was never attained due to the existence of these facilities within the time-sharing network and due to severe constraints on the time available for program development. Happily these constraints were self-imposed due to the large number of credit applications being processed.

At this point I believe there is some value in reflecting upon the preconceptions which existed at the time a decision was taken to develop an in-house evaluation capability. Table 7 illustrates a number of decision criteria and the degree of importance placed upon them. The letter "B" (Before) represents the relative degree of importance placed on various aspects of the computing environment at the inception of the development of the project evaluation capability. The letter "A" (After) represents the writer's current thinking on the subject. In fairness it should be noted that the original design parameters were subjected to several changes in scope as the evaluation capability developed.

As indicated in Table 7, an interactive capability was deemed "critical" and indeed this has continued to be a prime requisite for efficient and timely project appraisal.

The question of support staff availability was considered initially and was thought to be "desirable". In retrospect it has proven to be a "critical" ingredient. On a number of occasions, usually under conditions of duress, recourse to the time-sharing company's support staff has been invaluable.

At the outset, an extensive program library containing statistical, business and scientific programs was thought to be a "critical" consideration. In fact, little use is made of computational packages and in retrospect this system characteristic is rated "desirable".

The question of batch processing is interesting in that it was considered "critical" at the outset but has proved to be totally "not important". This misconception was due to a number of factors such as the over-estimation of interactive computing costs, underestimation of the efficiencies of APL in business applications, and the initial impression that programs could be run at a leisurely pace with 12 to 24 hour delays. It is not expected that the batch processing characteristic will become any more important in the future since the necessity for frequent program modifications coupled with severe time constraints between the receipt of data and the necessity for making a decision will likely continue to override any incremental cost considerations.

Another decision criterion was the capability of directing output to a highspeed printer. This was originally deemed "important" and in fact has proven to be so.

The existence of an extensive communications network was not even considered and by default is indicated as being "not important" in Table 7. However, as the evaluation capability matured, frequent use was made of the communications network in order to run simulations from various cities in North America. Alternately, the highspeed printing of project files in Calgary and in Vancouver via the communications network has proven an economical and effective way of disseminating information on short notice.

The last two items in Table 7 concern formatting aids and filing aids. These were initially considered to be "desirable" and "not important" respectively, however, they have proved to be of "critical" importance in efficiently evaluating mining projects when time considerations require rapid report generation and self-documenting files. Specifically, the formatting and filing aids referred to, form part of the AIDS simulation system offered by I.P. Sharp Associates. This "simulation and report generating package" was primarily responsible for the short time lag between the initiation of the project evaluation capability and its implementation on real projects.

Three Concluding Notes

An effective project evaluation capability permits a truly quantitative analysis of a project and its sensitivity to variations in input parameters such as metal prices, capital costs, capital cost overruns, operating costs, reserve grades, product blends (for coal and asbestos deposits), mill recoveries, smelter efficiencies, transportation and marketing costs.

The detailed business of formatting a logical cash flow and debt servicing pro-forma and the ensuing coding of the line-to-line and year-to-year mathematical relationships between input variables and derived results forces a more penetrating analysis of the probable behaviour of mining projects under a variety of conditions.

While designed primarily as a "passive" analytical tool, the project evaluation capability has been instrumental in generating new business.

```
FILE:           TERM                                          TABLE 1
CASE 0 : BASE CASE
YEARS 1 THRU 10

   1 PERIOD □                        0      0      0      0      0      0      0      0      0      0
   2 INTEREST RATE □             .000   .000   .000   .000   .000   .000   .000   .000   .000   .000
   3 TOTAL FUNDS REQUIRED □          0      0      0      0      0      0      0      0      0      0
   4 DEBT/EQUITY RATIO □           .00    .00    .00    .00    .00    .00    .00    .00    .00    .00
   5 OPERATING PROFIT □              0      0      0      0      0      0      0      0      0      0
   6 DEPRECIATION                    0      0      0      0      0      0      0      0      0      0
   7 DEBT INTEREST(°/°)              0      0      0      0      0      0      0      0      0      0
   8 PRE-TAX INCOME                  0      0      0      0      0      0      0      0      0      0
   9 C C A                           0      0      0      0      0      0      0      0      0      0
  10 TAXABLE INCOME                  0      0      0      0      0      0      0      0      0      0
  11 DEFERRED TAX                    0      0      0      0      0      0      0      0      0      0
  12 TAX PAID (CURRENT)              0      0      0      0      0      0      0      0      0      0
  13 TAX PAID (FROM DEFERRED)        0      0      0      0      0      0      0      0      0      0
  14 TOTAL TAX PAID                  0      0      0      0      0      0      0      0      0      0
  15 CUM TOTAL TAX PAID              0      0      0      0      0      0      0      0      0      0
  16 NET PROFIT                      0      0      0      0      0      0      0      0      0      0
  17 DEBT O/S BEGIN YEAR             0      0      0      0      0      0      0      0      0      0
  18 C.F. GENERATED                  0      0      0      0      0      0      0      0      0      0
  19 DEBT O/S END YEAR               0      0      0      0      0      0      0      0      0      0
  20 NET C.F. TO PROJECT             0      0      0      0      0      0      0      0      0      0
  21 CUM NET C.F. TO PROJECT         0      0      0      0      0      0      0      0      0      0
  22 °/° C.F. TO DEBT □            .00    .00    .00    .00    .00    .00    .00    .00    .00    .00
  23 R.O.R. ON EQUITY               .0     .0     .0     .0     .0     .0     .0     .0     .0     .0
  24 P.V. OF N.C.F. AT 12°/°         0      0      0      0      0      0      0      0      0      0
  25 GOVT TAX TAKE(PV 8°/°)          0      0      0      0      0      0      0      0      0      0
  26 TOTAL INT PAID                  0      0      0      0      0      0      0      0      0      0
  27 R.O.R. ON TOTAL FUNDS          .0     .0     .0     .0     .0     .0     .0     .0     .0     .0
```

TABLE 2

```
FILE:          TERM
CASE 0 : BASE CASE
YEARS 1 THRU 10

 1 PERIOD ☐                             1      2      3      4      5      6      7      8      9     10
 2 INTEREST RATE ☐                   .100   .100   .100   .100   .100   .100   .100   .100   .100   .100
 3 TOTAL FUNDS REQUIRED ☐           10000      0      0      0      0      0      0      0      0      0
 4 DEBT/EQUITY RATIO ☐                .75    .75    .75    .75    .75    .75    .75    .75    .75    .75
 5 OPERATING PROFIT ☐                2250   2250   2250   2250   2250   2250   2250   2250   2250   2250
 6 DEPRECIATION                         0      0      0      0      0      0      0      0      0      0
 7 DEBT INTEREST(°/°)                   0      0      0      0      0      0      0      0      0      0
 8 PRE-TAX INCOME                       0      0      0      0      0      0      0      0      0      0
 9 C C A                                0      0      0      0      0      0      0      0      0      0
10 TAXABLE INCOME                       0      0      0      0      0      0      0      0      0      0
11 DEFERRED TAX                         0      0      0      0      0      0      0      0      0      0
12 TAX PAID (CURRENT)                   0      0      0      0      0      0      0      0      0      0
13 TAX PAID (FROM DEFERRED)             0      0      0      0      0      0      0      0      0      0
14 TOTAL TAX PAID                       0      0      0      0      0      0      0      0      0      0
15 CUM TOTAL TAX PAID                   0      0      0      0      0      0      0      0      0      0
16 NET PROFIT                           0      0      0      0      0      0      0      0      0      0
17 DEBT O/S BEGIN YEAR                  0      0      0      0      0      0      0      0      0      0
18 C.F. GENERATED                       0      0      0      0      0      0      0      0      0      0
19 DEBT O/S END YEAR                    0      0      0      0      0      0      0      0      0      0
20 NET C.F. TO PROJECT                  0      0      0      0      0      0      0      0      0      0
21 CUM NET C.F. TO PROJECT              0      0      0      0      0      0      0      0      0      0
22 °/° C.F. TO DEBT ☐                 .85    .85    .85    .85    .85    .85    .85    .85    .85    .85
23 R.O.R. ON EQUITY                    .0     .0     .0     .0     .0     .0     .0     .0     .0     .0
24 P.V. OF N.C.F. AT 12°/°              0      0      0      0      0      0      0      0      0      0
25 GOVT TAX TAKE(PV 8°/°)               0      0      0      0      0      0      0      0      0      0
26 TOTAL INT PAID                       0      0      0      0      0      0      0      0      0      0
27 R.O.R. ON TOTAL FUNDS               .0     .0     .0     .0     .0     .0     .0     .0     .0     .0
```

TABLE 3

```
FILE:        TERM
CASE 0 : BASE CASE
YEARS 1 THRU 10

 1 PERIOD □                      1      2      3      4      5      6      7      8      9     10
 2 INTEREST RATE  □           .100   .100   .100   .100   .100   .100   .100   .100   .100   .100
 3 TOTAL FUNDS REQUIRED  □   10000      0      0      0      0      0      0      0      0      0
 4 DEBT/EQUITY RATIO  □        .75    .75    .75    .75    .75    .75    .75    .75    .75    .75
 5 OPERATING PROFIT  □        2250   2250   2250   2250   2250   2250   2250   2250   2250   2250
 6 DEPRECIATION               1000   1000   1000   1000   1000   1000   1000   1000   1000   1000
 7 DEBT INTEREST(∘/∘)          750    623    484    334    171     18      0      0      0      0
 8 PRE-TAX INCOME              500    628    766    916   1079   1232   1250   1250   1250   1250
 9 C C A                      1500   1628   1766   1916   1510    504    353    247    173    121
10 TAXABLE INCOME                0      0      0      0    569   1728   1897   2003   2077   2129
11 DEFERRED TAX                240    301    368    440    245      0      0      0      0      0
12 TAX PAID (CURRENT)            0      0      0      0    273    829    911    961    997   1022
13 TAX PAID (FROM DEFERRED)      0      0      0      0      0    238    311    361    397    286
14 TOTAL TAX PAID                0      0      0      0    273   1067   1221   1323   1394   1308
15 CUM TOTAL TAX PAID            0      0      0      0    273   1340   2562   3884   5278   6587
16 NET PROFIT                  500    628    766    916    806    165     29    ¯73   ¯144    ¯58
17 DEBT O/S BEGIN YEAR        7500   6225   4842   3341   1712    177      0      0      0      0
18 C.F. GENERATED             1500   1628   1766   1916   1806   1165   1029    927    856    942
19 DEBT O/S END YEAR          6225   4842   3341   1712    177      0      0      0      0      0
20 NET C.F. TO PROJECT         225    244    265    287    271    988   1029    927    856    942
21 CUM NET C.F. TO PROJECT     225    469    734   1021   1292   2280   3309   4236   5092   6034
22 ∘/∘ C.F. TO DEBT  □         .85    .85    .85    .85    .85    .85    .85    .85    .85    .85
23 R.O.R. ON EQUITY           14.6     .0     .0     .0     .0     .0     .0     .0     .0     .0
24 P.V. OF N.C.F. AT 12∘/∘    2873      0      0      0      0      0      0      0      0      0
25 GOVT TAX TAKE(PV 8∘/∘)     4542      0      0      0      0      0      0      0      0      0
26 TOTAL INT PAID             2380      0      0      0      0      0      0      0      0      0
27 R.O.R. ON TOTAL FUNDS       6.8     .0     .0     .0     .0     .0     .0     .0     .0     .0
```

TABLE 4

```
⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
                                                          ∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇∇
                                                          CAPITAL PROJECT MODEL
                                                          ∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆∆
                                     INTEREST RATE VARIATION SERIES          CASE 0  :  BASECASE NO. 1
⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕

                                   1         2         3         4         5         6         7         8         9        10

<<INPUT PARAMETERS>>
                            ========  ========  ========  ========  ========  ========  ========  ========  ========  ========
INTEREST RATE ⎕                0.100     0.100     0.100     0.100     0.100     0.100     0.100     0.100     0.100     0.100
TOTAL FUNDS REQUIRED ⎕        10,000         0         0         0         0         0         0         0         0         0
DEBT/EQUITY RATIO ⎕             0.75      0.75      0.75      0.75      0.75      0.75      0.75      0.75      0.75      0.75
OPERATING PROFIT ⎕             2,250     2,250     2,250     2,250     2,250     2,250     2,250     2,250     2,250     2,250
∘/∘ C.F. TO DEBT ⎕              0.85      0.85      0.85      0.85      0.85      0.85      0.85      0.85      0.85      0.85
                            ========  ========  ========  ========  ========  ========  ========  ========  ========  ========

<<OUTPUT RESULTS>>
                            ========  ========  ========  ========  ========  ========  ========  ========  ========  ========
DEPRECIATION                   1,000     1,000     1,000     1,000     1,000     1,000     1,000     1,000     1,000     1,000
DEBT INTEREST(∘/∘)               750       623       484       334       171        18         0         0         0         0
PRE-TAX INCOME                   500       628       766       916     1,079     1,232     1,250     1,250     1,250     1,250
C C A                          1,500     1,628     1,766     1,916     1,510       504       353       247       173       121
TAXABLE INCOME                     0         0         0         0       569     1,728     1,897     2,003     2,077     2,129
DEFERRED TAX                     240       301       368       440       245         0         0         0         0         0
TAX PAID (CURRENT)                 0         0         0         0       273       829       911       961       997     1,022
TAX PAID (FROM DEFERRED)           0         0         0         0         0       238       311       361       397       286
TOTAL TAX PAID                     0         0         0         0       273     1,067     1,221     1,323     1,394     1,308
CUM TOTAL TAX PAID                 0         0         0         0       273     1,340     2,562     3,884     5,278     6,587
NET PROFIT                       500       628       766       916       806       165        29       ¯73      ¯144       ¯58

DEBT O/S BEGIN YEAR            7,500     6,225     4,842     3,341     1,712       177         0         0         0         0
C.F. GENERATED                 1,500     1,628     1,766     1,916     1,806     1,165     1,029       927       856       942
DEBT O/S END YEAR              6,225     4,842     3,341     1,712       177         0         0         0         0         0

NET C.F. TO PROJECT              225       244       265       287       271       988     1,029       927       856       942
CUM NET C.F. TO PROJECT          225       469       734     1,021     1,292     2,280     3,309     4,236     5,092     6,034

R.O.R. ON EQUITY                14.6       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0
R.O.R. ON TOTAL FUNDS            6.8       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0       0.0

P.V. OF N.C.F. AT 12∘/∘        2,873         0         0         0         0         0         0         0         0         0
GOVT TAX TAKE(PV 8∘/∘)         4,542         0         0         0         0         0         0         0         0         0
TOTAL INT PAID                 2,380         0         0         0         0         0         0         0         0         0
                            ========  ========  ========  ========  ========  ========  ========  ========  ========  ========
                            ========  ========  ========  ========  ========  ========  ========  ========  ========  ========
CASE 0 : U.S COAL PROJECT                              TABLE 5
YEARS 1 THRU 10

  1 PERIOD                          1       2       3       4       5       6       7       8       9      10
  2 YEAR                         1978    1979    1980    1981    1982    1983    1984    1985    1986    1987
101 RAW TONS MINED                200     300     450     450     450     450     450     450     450     450
102 COST RAW MINED/TON          14.00   13.50   13.50   13.50   13.50   13.50   13.50   13.50   13.50   13.50
103 YIELD                         .87     .87     .87     .87     .87     .87     .87     .87     .87     .87
104 CONTRACT TONS MINED           200     250     250     250     250     250     250     250     250     250
105 COST / CONTRACT TON         15.00   15.00   15.00   15.00   15.00   15.00   15.00   15.00   15.00   15.00
106 YIELD                         .87     .87     .87     .87     .87     .87     .87     .87     .87     .87
107 TONS SOLD                     348     479     609     609     609     609     609     609     609     609
108 MINING COST / TON SOLD      16.67   16.30   16.13   16.13   16.13   16.13   16.13   16.13   16.13   16.13
109 PERCENT WASHED                .00     .00    1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00
110 WASHING COST/ CLEAN TON      1.02    1.02    1.02    1.02    1.02    1.02    1.02    1.02    1.02    1.02
  4 COAL PRICE ($/TON)          25.00   25.00   30.75   30.75   30.75   30.75   30.75   30.75   30.75   30.75
  6 +GROSS SALES ($000'S)        8700   11963   18727   18727   18727   18727   18727   18727   18727   18727
  7 OPERATING COST ($000'S)      5800    7800   10446   10446   10446   10446   10446   10446   10446   10446
  8 S.G.AND A ($000'S)            783    1077    1545    1545    1545    1545    1545    1545    1545    1545
  9 ROYALTIES ($000'S)            326     598     936     936     936     936     936     936     936     936
 10 OVERRIDING R'YLTY($000'S)     150     150     150     150     152     305     305     305     305     305
 11 DEPRECIATION ($000'S)         200     560     888     990     832     706     605     524     459     407
 12 STATE TAX                     335     461     721     721     721     721     721     721     721     721
 13 -TOTAL COST OF SALES         7594   10645   14687   14789   14633   14659   14558   14477   14412   14361
 14 =OPERATING INCOME($000'S)    1106    1317    4040    3938    4093    4068    4169    4250    4314    4366
 15 -INTEREST EXPENSE 1           690     690     690     640     540     440     340     240     140      90
 16 -INTEREST EXPENSE 2           130     310     410     410     310     210     110      10       0       0
 17 =INCOME AFTER INT($000'S)     286     317    2940    2888    3243    3418    3719    4000    4174    4276
 18 -DEPLETION ALLOWANCE          143     159    1470    1444    1622    1709    1779    1779    1779    1779
 19 =TAXABLE INCOME               143     159    1470    1444    1622    1709    1940    2221    2395    2497
 20 TAX AT 48°/°($000'S)           69      76     706     693     778     820     931    1066    1150    1199
 21 -TAX AFTER I.T.C.              34      38     3 3     618     758     800     911    1046    1130    1179
 22 =NET INCOME                   252     279    2587    2269    2485    2617    2808    2954    3045    3098
 23 CASH FLOW                     452     839    3475    3260    3317    3323    3412    3478    3504    3505
 25 WORKING CAPITAL REQU'MENT     500       0       0       0       0       0       0       0       0       0
 51 PLANT AND EQUIPMENT          1500    1800    1000       0       0       0       0       0       0       0
 52 EQUIPMENT REPLACEMENT         100     200     200     200     200     200     200     200     200     200
 53 NON-ITC CAP. EXPEND.          400     200     200       0       0       0       0       0       0       0
 54 PURCHASE LOAN (1)            6000       0       0       0       0       0       0       0       0       0
112 CAP. EQUIP. LOAN (2)         1300    1800    1000       0       0       0       0       0       0       0
 27 WORKING CAPITAL INJECTION     900       0       0       0       0       0       0       0       0    -900
113 TOTAL LOANS                  8200    1800    1000       0       0       0       0       0       0    -900
 28 PAYMENTS LOAN 1 ($000'S)        0       0     500    1000    1000    1000    1000    1000     500       0
 29 PAYMENTS LOAN 2 ($000'S)        0       0       0    1000    1000    1000    1000     300       0       0
 30 TOTAL LOAN PAYMENTS             0       0     500    2000    2000    2000    2000    1300     500       0
 31 CURRENT AVAIL FOR PMTS        152     439    3075    3060    3117    3123    3212    3278    3304    2405
 32 NET C.F. TO PROJECT           152     439    2575    1060    1117    1123    1212    1978    2804    2405
 33 CUM NET C.F. TO PROJECT       152     591    3166    4226    5343    6466    7678    9656   12459   14864
 62 P.V. OF NET C.F. (12°/°)     7327       0       0       0       0       0       0       0       0       0
 63 R.O.R. ON FUNDS EMPLOYED     20.5      .0      .0      .0      .0      .0      .0      .0      .0      .0
```

TABLE 6

LOAN AMOUNT	COMPUTER COSTS (±20%)
$40,000,000	$1,700
$70,000,000	$3,800
$85,000,000	$2,200
$10,000,000	$2,100
$350,000,000	$2,500
$60,000,000	$300

TABLE 7

SUBJECTIVE ASSESSMENT OF THE IMPORTANCE OF TIMESHARING SYSTEM CHARACTERISTICS - BEFORE AND AFTER

B - INITIAL (JULY, 1976) A - ACTUAL (SEPTEMBER, 1978)

DECISION CRITERIA	DEGREE OF IMPORTANCE			
	Critical	Important	Desirable	Not Important
INTERACTIVE	B,A			
SUPPORT STAFF	A		B	
PROGRAMME LIBRARY	B		A	
BATCH PROCESSING	B			A
HIGH-SPEED PRINTER		B,A		
COMMUNICATIONS NETWORK	A			B
FORMATTING AIDS	A		B	
FILING AIDS	A			B

<u>FIGURE 1</u>

```
      ∇CAPMODEL[⎕]∇
    ∇ CAPMODEL
[1]   READID 1 2 3 4 5 22 ⍝   FUNCTION <<READID>> READS INPUT VARIABLES FROM DATA MATRIX
[2]   CCAPOOL←0 ◇ I←1 ⍝ SET COUNTERS
[3]   A6←A7←A8←A9←A10←A11←A12←A13←A14←A15←A16←A17←A18←A19←A20←A21←A23←A24←A25←A26←A27←(⍴A1)⍴0 ⍝ ZERO OUT OUTPUT LINES
[4]   A6←(⍴A1)⍴(A3[1]÷⍴A1) ⍝ DEPRECIATION - STRAIGHT LINE OVER ⍴A1 YEARS.
[5]   CCASTRING←(0.3×A3[1]),(0.3×A3[1])×0.7*⍳9 ⍝   CCA ASSUMING 30∘/∘ RATE.
[6]   A7[1]←A2[1]×A3[1]×A4[1] ◇ →L2 ⍝   CALCULATE INTEREST FOR YEAR 1
[7]  L1:A7[I]←A2[I]×A19[I-1] ⍝   CALCULATE INTEREST FOR YEAR I
[8]  L2:A8[I]←A5[I]-A6[I]+A7[I] ⍝   PRE-TAX INCOME
[9]   CCAPOOL←CCAPOOL+CCASTRING[I] ◇ A9[I]←0⌈(A5[I]-A7[I])⌊CCAPOOL ◇ CCAPOOL←0⌈CCAPOOL-A9[I] ⍝ CALC. CCA AND ADJUST CCAPOOL
[10]  A10[I]←A5[I]-A9[I]+A7[I] ◇ A11[I]←0⌈0.48×A9[I]-A6[I] ◇ A12[I]←0.48×A10[I] ⍝  TAX. INC./DEF. TAX / TAX PAID(CURRENT)
[11]  A13[I]←0⌈(0.48×A6[I]-A9[I])⌊(0⌈(+/A11[⍳I])-(+/A13[⍳I])) ⍝   CALC TAX PAID (FROM DEFERRED)
[12]  A14[I]←A12[I]+A13[I] ◇ A15[I]←+/A14[⍳I] ◇ A16[I]←A8[I]-A14[I] ⍝ TOT TAX PAID / CUM TOT TAX PAID / NET PROFIT
[13]  A17[I]←A19[1⌈I-1]+(A3[I]×A4[I]) ◇ A18[I]←A5[I]-A7[I]+A14[I] ⍝ O/S BEGIN YEAR  /  C.F. GENERATED
[14]  A19[I]←0⌈A17[I]-A22[I]×A18[I] ⍝ O/S END OF YEAR
[15]  A20[I]←0⌈((1-A22[I])×A18[I])+|0⌊A17[I]-(A22[I]×A18[I]) ◇ I←I+1 ⍝ N.C.F. TO PROJECT  /  INCREASE COUNTER
[16]  →(I≤⍴A1)⍴L1 ⍝ RETURN TO L1 IN ORDER TO PROCESS NEXT COLUMN
[17]  A21←+\A20 ◇ A26[1]←+/A7 ⍝  CUM. N.C.F. TO PROJECT   /   TOTAL INT. PAID
[18]  GET 'ROR' ⍝ GET FUNCTION <<ROR>> FROM MODELS FILE
[19]  A23[1]←(A3[1]×1-A4[1]) ROR A20 ⍝ CALC R.O.R. ON EQUITY
[20]  A27[1]←A3[1] ROR A18 ⍝ CALC. R.O.R. ON TOTAL FUNDS EMPLOYED
[21]  A24[1]←+/A20÷1.12*⍳⍴A1 ⍝ P.V.  OF N.C.F. AT 12∘/∘
[22]  A25[1]←+/(A14+0.48×A7)÷1.08*⍳⍴A1 ⍝ P.V. OF GOV'T TAKE AT 8∘/∘
[23]  LIST← 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 23 24 25 26 27 ⍝ ARGUMENT FOR FUNCTION <<STORE>>
[24]  LIST STORE(21,⍴A1)⍴A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16,A17,A18,A19,A20,A21,A23,A24,A25,A26,A27 ⍝STORE OUTPUT
[25]  WRITE ⍝ DATA TO FILE USING FUNCTION <<WRITE>>
    ∇
```

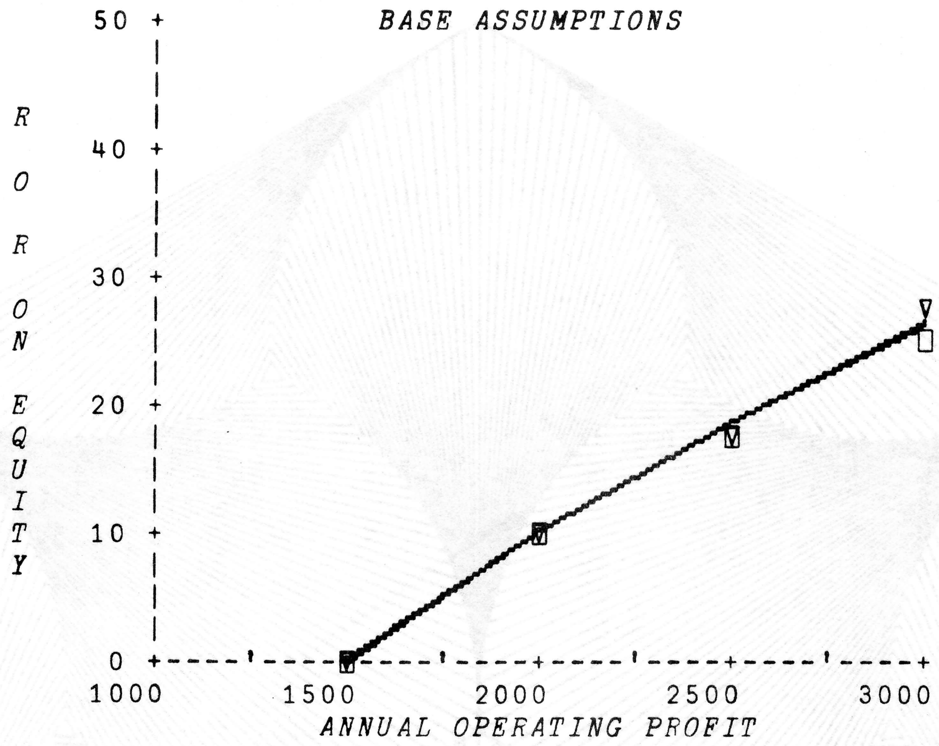

BASE ASSUMPTIONS

FIGURE 2

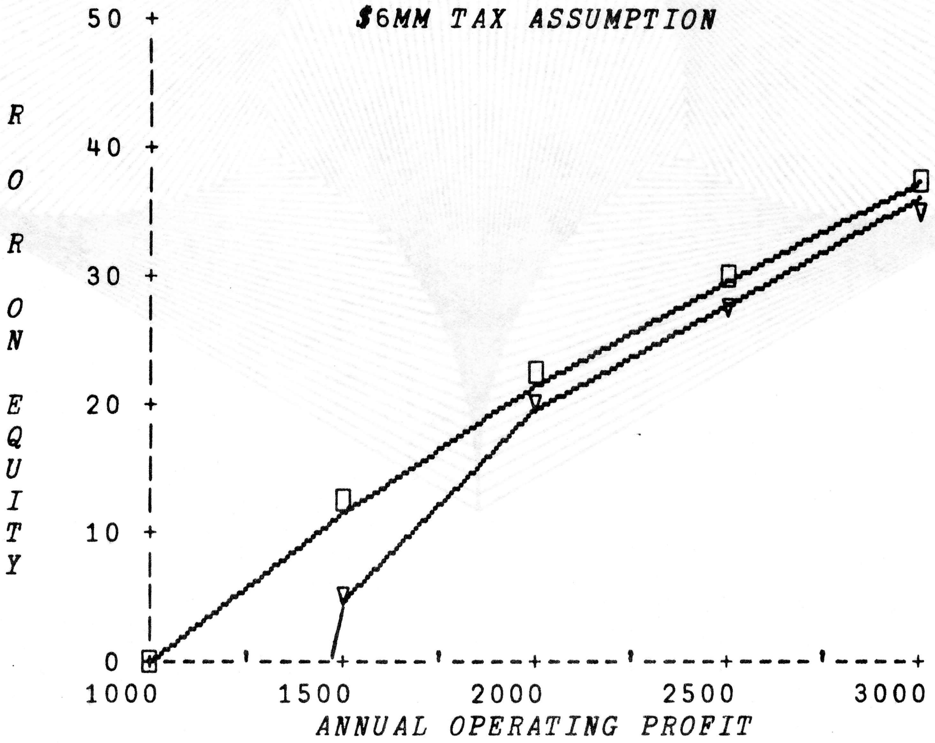
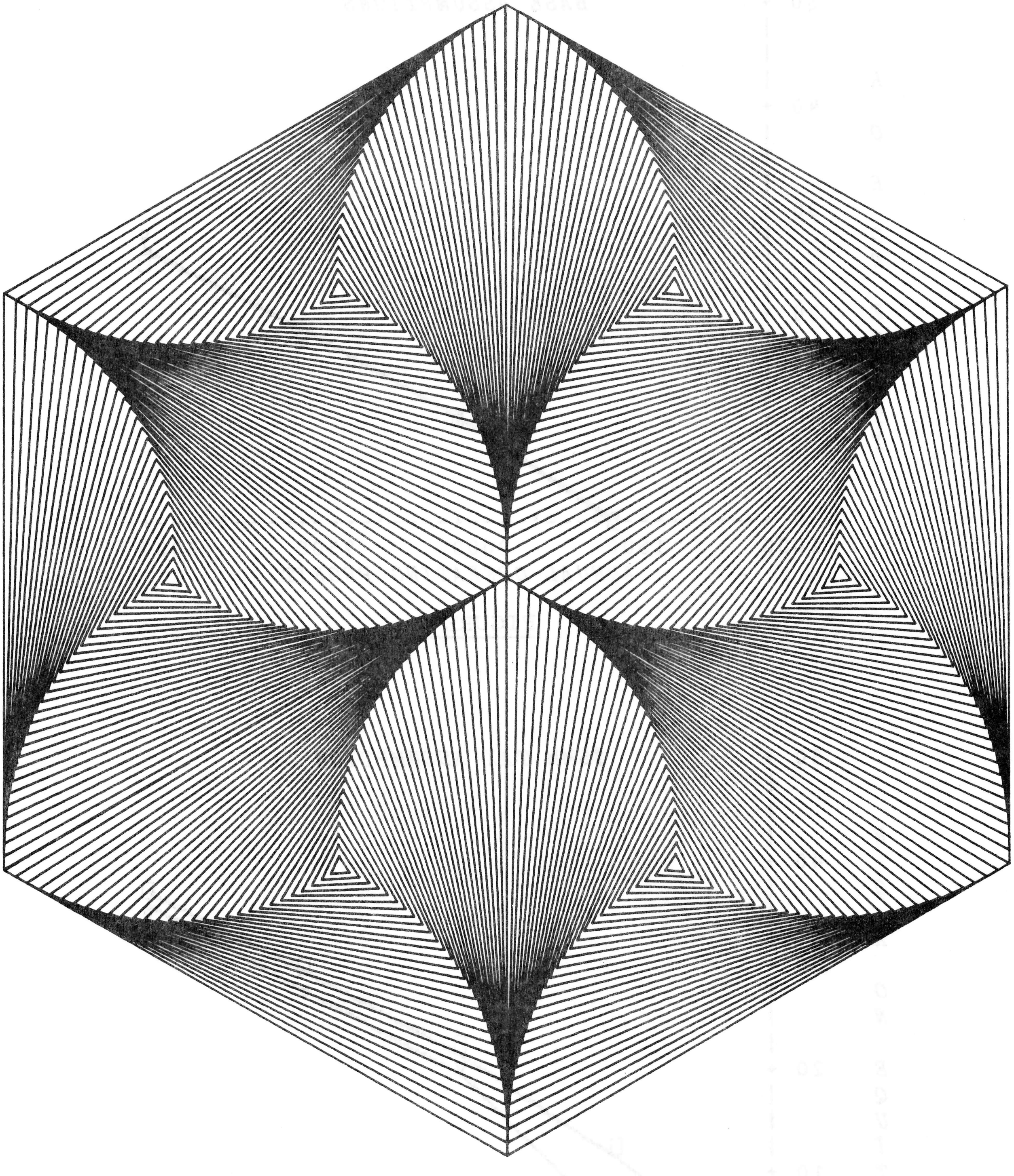

$6MM TAX ASSUMPTION

FIGURE 3



FINANCIAL PLANNING IN INTERNATIONAL BANKING

W. Lee Bettes
First National Bank Of Chicago
Chicago, Illinois

Since the emergence of modern commercial banking, two separate groups of voices have rung out with the same battle cry. "Planning", cheered the senior management of most banks, "will allow us to maximize income, minimize cost, effectively allocate resources and monitor and control performance."

"Planning", cursed the administrative staffs of those banks, "will bury us. We must complete an unending stream of planning forms. We must even plan how much time we plan to spend planning. And who cares?"

No one has all of the answers on how to totally reconcile these conflicting views. It is, however, the intent of this paper to share certain guidelines that have been developed at the First National Bank of Chicago over the last decade. These guidelines will tend to be pragmatic for the ease of presentation. (The terms planner and user will be used interchangeably to represent one who plans with the help of a computer.)

What is a bank?

A bank is a financial intermediary which earns income by buying and selling money. Exhibit (1) shows a simplified view of a bank. Funds flow into a bank by people making deposits or by paying off loans. Funds flow from the bank by depositors withdrawing deposits or borrowing money and by the bank paying its expenses (interest, salaries, etc.) and dividends.

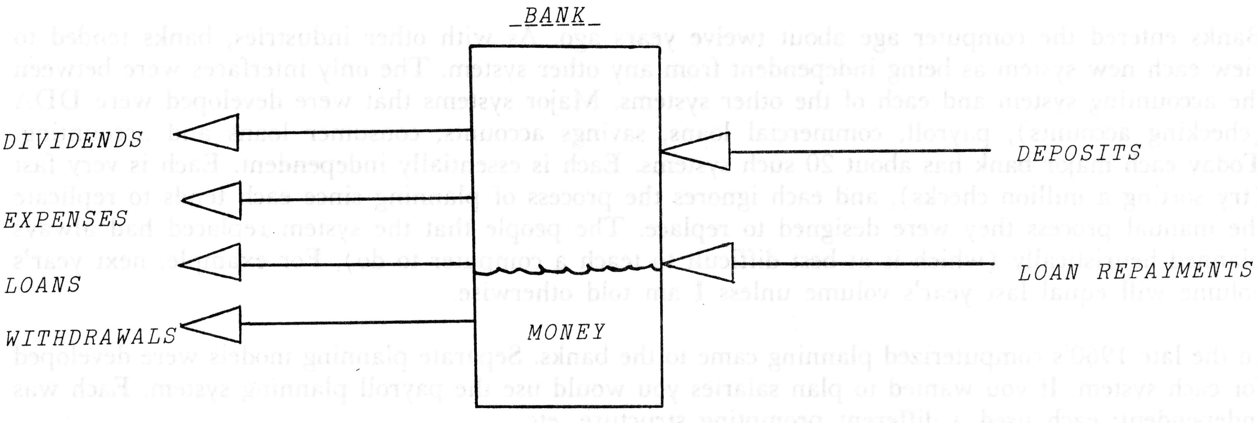


EXHIBIT (1)

It is the responsibility of bank management to regulate the flow of those funds so that all deposits are safe, individuals can borrow money and shareholders receive a fair return on their investment.

Sounds easy and perhaps it was in the days of Shylock, but not so today. Banks must now manage tens of billions of dollars. For large international banks, those dollars may not necessarily be dollars but may be a mix of dozens of different currencies. Loans can be for any length and size (leases if you prefer). Deposits types are available by the hundreds. There are thousands of employees to pay, train and support. Add a few million customers, spread the bank's operation around the world, mix in vocal consumer groups and add more governmental regulations than most any other industry and instead of six arrows on the chart a few hundred thousand arrows might be more appropriate. That is a bank, General Custer.

So why plan?

Never have I heard a good answer to this. Perhaps the best answer is that planning is the business of management. If you do not plan you are needlessly subjecting your bank to risk and ultimately you will be replaced by those who will plan. (Translation-Banks and bankers that do not plan may lose big.) Excluding their own well-being, management's desire to plan usually is based on a desire to give direction to the bank, and once the plans have been made, to monitor and control the bank's progress in achieving those plans. Specific areas of interest to management are generally:

1) Profit maximization subject to certain risk constraints.

2) Resource allocation - this varies from who gets a loan to which area of the bank gets programming support. (We have over 180 programmers, so this in itself is no simple task.)

3) Cost minimization - no comment needed.

A little background

Banks entered the computer age about twelve years ago. As with other industries, banks tended to view each new system as being independent from any other system. The only interfaces were between the accounting system and each of the other systems. Major systems that were developed were DDA (checking accounts), payroll, commercial loans, savings accounts, consumer loans and accounting. Today each major bank has about 20 such systems. Each is essentially independent. Each is very fast (try sorting a million checks), and each ignores the process of planning since each tends to replicate the manual process they were designed to replace. The people that the system replaced had always planned heuristically (which is at best difficult to teach a computer to do). For example, next year's volume will equal last year's volume unless I am told otherwise.

In the late 1960's computerized planning came to the banks. Separate planning models were developed for each system. If you wanted to plan salaries you would use the payroll planning system. Each was independent; each used a different prompting structure, etc.

So what's the problem?

This seems great except for four issues:

1) Most models were designed by accountants (or management scientists who thought they were accountants) who were more concerned with line items, control totals and good-looking reports. Little emphasis was placed on **financial** planning. There will be more on this later.

2) The existing models didn't supply bank management with the global view of the bank. Each model produced a good plan for its own area but those plans were not merged into one overall plan.

3) The existing models were difficult to use from a clerical (human engineering) viewpoint. For example, a commercial loan planner, who might want to use the commercial loan model, might have a computer session as follows (with translations in parentheses):

```
Computer: \ * c                    (Speak to me.)

User: \ . proc. loan \\            (I am ready to plan commercial loans.)

Computer: \ . file                 (Which files do you wish to use?)

User: \ . 27849. *                 (The same ones I always use.)

Computer: \ . loans                (Enter your loan volume for next year. Please include
                                   existing loans that will not have matured.)

User: 689 714 713 724 720 740 760 770 780 790 800

Computer: / . * * File error.      (You only gave me 11 numbers and I have inadvertent-
                                   ly destroyed the data base.)
```

4) The existing models from a programming standpoint were difficult to maintain. Bank building may only change every 75 years but the internal organizational structure of a bank is continually changing. The development of financial models was usually not done using the standards employed by the rest of the data processing staff of the bank. Each reorganization or accounting change may require 10 or more models to be changed.

What's the answer?

There is no one answer. It is the right of each financial institution to make its own mistakes. However, we have developed the following guidelines over the years for our institution. System personnel should strive to achieve these goals. Planners should demand them.

Build financial models not accounting models. Bank planning models tend to be accounting models which concentrate on calculating individual line items of a balance sheet, income statement, and sources and uses of cash statement. Little effort has been expended at considering each line item as part of a set of equations that should be solved simultaneously. Integrated into this must be the dynamic relationship of the asset and liability mix and the incorporation of rate/volume sensitivity analysis. The object of the model must be to entice planners into asking "what if?".

In support of the "what if" type planning, each department/group/unit/etc. that is required to submit a plan should be required to submit at least three plans. Why three plans? The three was arbitrary but the idea of multiple plans is that a large portion of any bank's profitability and liquidity is based on what the world/national economy is going to do over the planning period. For example, the central staff (corporate economist perhaps) should develop three scenarios for the planning period such as:

(a) Interest rates will remain unchanged. Gross National Product (GNP) will increase 7%. The U.S. dollar will remain stable.

(b) Interest rates will drop 1 1/2%. GNP will rise 5%. The U.S. dollar will devalue 4%.

(c) Interest rates will rise 1%. GNP will rise 8%. The U.S. dollar will evaluate 5%.

Not shown but of interest could be projections on legal changes, competition and inflation. With such information, planners can concentrate on those factors that are under their control (i.e., marketing, advertising, manpower, etc.). Plans would then reflect what each unit plans to do given a set of economic forecasts.

Planners must be told how well they have done. The planning system should be integrated into the accounting systems so that as the plan becomes reality, the planned data (whichever scenario came true) is compared against the actual results. People like to be right and where necessary can learn from their mistakes. Financial compensation for planners should be tied to their planned versus actual results but only to the extent that the results were under their control.

The planning process must be iterative. Definitive feedback loops must be built into the planning cycle. Senior management must, from the outset, give economic projections and management guidelines (e.g., where to concentrate resources). Planners should complete and submit their plans. Senior management should then review those plans and return those plans which, for whatever reason, are unacceptable for replanning. Most likely there will be 3 or 4 iterations. Those iterations are a burden, but at the conclusion, those plans cross the magical boundary from plans to goals since both parties have accepted them. (I have heard many people complain of planning but never of goal setting.)

The planning model must reflect the nature of the bank's business. This is perhaps the most difficult area to define. Each bank will be different but I'll include an example of what might be correct for a large international bank that makes loans in 5 different currencies and must offset each loan with a purchase of an equal amount of money.

Computer: Which currency will this loan be in?

User: Help

Computer: You may choose one of the following currencies:

A) US

B) Canadian

C) English

D) Swiss

E) German

User: D

Computer: Enter company name, loan amount in Swiss francs and length in months, and interest rate.

User: ABC INC 8000000 24 .08

The computer would then add this loan to the plan (after the francs were converted to the currency used by the bank's accounting system) and add a corresponding entry in the plan noting the need to acquire enough francs for the loan. This example is trivial but the point is that the computer should automatically take care of any process that can be computerized such as foreign exchange, loan funding, etc., and leave judgmental factors to the planners.

The planning model and process should include contingency analysis. The planning process as referenced earlier concentrates on likely economic scenarios. Contingency analysis on the other hand is concerned with radically altered economic scenarios that have a low probability but potentially high impact. It is suggested that each year your bank evaluate what it (the bank) would do given that a nightmarish economic projection comes true. For example, one such projection could be that the prime interest rate would rise to 14% and severe disintermediation would occur simultaneously.

Given such a projection, one would hope that the bank would radically alter its asset/liability mix, marketing strategy and loan policies. The benefit of such an exercise (contingency analysis) would be twofold.

(a) Should such a scenario come true, bank management would have a strategy in place that was developed in a non-crisis environment.

(b) New or better insights would be developed into the financial structure of the bank. More creative thinking would be required of management/planners. Just the educational benefits of once a year developing and selecting difficult courses of action should be worth the effort.

Contingency analysis is widely used by the military. Their belief is that when a war occurs it is too late to start planning. Our battle fields may not be physical, but those of you who were involved in the credit crunch of 1974 know that our financial markets can at least be tense.

Have the computer serve the planner as an intelligent, humble servant. This is the area where many financial models have failed.

(a) All input should be fully edit checked by the computer. The goal is to have the computer check each input, and politely (but firmly) reject bad input. For example:

Computer: Enter probability

User: 25

Computer: 25 is unacceptable. The probability must be a number between zero (0) and one (1). For example, .55 is acceptable. Try again.

User: .25

(b) The computer must speak the user's native language. All prompting should make sense and should the user enter "HELP" the computer must be prepared to give a detailed description of what it is asking. The user's response should also be in the user's native language. I have tried it both ways and I assure you it is far easier to teach a computer the user's language than vice versa.

(c) The computer should do as much work as possible. This sounds trite but is often ignored. An example of this is our personnel planning system:

Computer: Enter employee's I.D.

User: 46625

Computer: John Q. Smith

Salary $16250

Review date February

Enter percent raise

User: .08

At that point, Smith's salary is calculated for the next year as are his taxes, healthcare and profit sharing. Depending on the situation, the model could also plan floor space needs, education, postage and travel. The planner could then move on to the next employee.

(d) All planning models for the corporate entity should appear to the planner as one model. A planner should be able to sign on to one computer system and do any type of planning that he needs (and is authorized) to do, even though there might be several different planning systems, each of which is written in a different computer language and uses different files. (Note to planners - your computer people will tell you that this is impossible. Simply smile and tell them that you have faith in their technical ability. They will find a way.)

In addition, all prompting (how the computer addresses the planner) should be consistent between different areas of the model. If a customer is referred to by account number in one area of the model, the same customer should not be referred to by name elsewhere in the model. (The planner should be allowed to initially choose which way, or have the ability to enter either throughout the model.) Cute or clever prompting should be avoided.

Computer reports must not look like computer reports. If planners are to use the model for "what if" type work, you want them to have reports that they can be proud to show management. It takes a little more programming effort but you will find that it's worth it. A good starting point is to exactly replicate the format of your annual report's financial statement. If that format (down to commas and dollar signs) is good enough for your shareholders, it's good enough for your planners.

Planning programmers must follow all technical standards used elsewhere in the bank. For years we, (I am one) the programmers, of planning models, have claimed that we should not have to follow all of the technical standards and guidelines required of other system personnel within our banks. Our arguments were good. "We are different! We don't need technical standards' we cried, making up in volume what we lacked in substance.

Times have changed. Planning is now one of the businesses of banking. As such, planning systems should be as reliable, dependable, and auditable as any check processor.

Conclusion

Planning for banks is at best difficult. Computer technology can, however, help. Planners must think of their work as financial in nature and should concentrate their efforts on asset/liability management, rate/volume sensitivity analysis, and resource allocation. Computer models should supply **all** of the help to this process that is possible.

Each planning cycle should be easier than the previous one. As this happens, with the aid of computers, planning as we know it today should die and be replaced with goal setting. Let us spend our time in banks contemplating what should happen next year, not how many planning forms have to be filled out.

In parting let me add a word of caution. The attributes listed above are ideals and should be used as goals. Banks are so complex today that the achievement of those goals will take years and once you have reached them you'll find that banking has changed as must your goals. Take heart - banks must always have computers and planners. The work is challenging. Good luck.

FORECASTING EUROPEAN AIR TRAVEL

Brian P. Loughran
Association of European Airlines
Brussels, Belgium

Introduction

A statistician is "a man who draws a straight line from unwarranted assumptions to a foregone conclusion". - J.W. Tukey.

Less than 75 years ago, in December 1903, Orville and Wilbur Wright made the first powered flight. Now modern aircraft, themselves longer than that first flight, link the whole planet, enabling the world's airlines to produce each year over one billion (i.e., one thousand, thousand million) seat kilometres. For the statistically minded, the compound annual growth rate since 1903 exceeds 33% per annum, and airlines are still expanding several times faster than general industry.

The initial climb of air transport depended most on engineering achievements - safer, faster, larger, more comfortable aircraft - achievements which, one must admit, were spurred on by otherwise regrettable phenomena such as war. But, as the industry matured, say from the introduction of jet services 20 years ago, commercial and economic trends have powered the industry ascent. The tradition of British understatement and the technical nature of this paper prevent my mentioning the role of airline management in the evolution of this great industry. Suffice to say that neither regulators, nor economists have seriously impaired progress, at least to date. My current theme, however, is that the increasing dominance of socio-economic forces have exposed the industry to a more turbulent environment and thus created need for forecasting methods which fully reflect this environment.

Throughout the airline industry, informal forecasting methods (group discussion, simple trend projection, or consulting oracles) are being complemented by analyses of how economic change affects air travel. The main European airlines - through the Association of European Airlines (AEA) - have, by pooling individual airline experience and data, constructed statistical models which relate air travel to economic trends. Hitherto, joint effort has mainly concentrated on short term forecasts (i.e., 5 years ahead) of scheduled passenger traffic between European countries. Since the technical methods may be of some interest, this paper describes:

(a) the choice of model forms;

(b) the computing system;

(c) the overall forecasting procedures;

(d) achieved accuracies and continuing research in model development.

Rational decision necessarily implies a guess about the future derived from the experience of the past. Specific analogies drawn from past experience - statistical models - are calibrated using implicit judgements, both historical and technical. Therefore we do not regard the models as rigid techniques to be applied blindly. The methodology though is indispensable for the systematic framework within which data and assumptions are recorded. Thus we can identify precisely why forecast errors arise. Since the study of errors is most valuable in adapting methods and evolving better forecasts, the AEA Forecasting Group combines:

(a) the responsibility of making forecasts with

(b) the responsibility of advancing forecasting techniques.

Thus progress and improved forecast accuracy are the results of co-operation among economic, traffic and technical experts from the national airlines of Europe.

Model Design

"Entities should not be multiplied unnecessarily." - Occam's Razor.

In 1974, the Economic Research Committee of the AEA agreed to assign a high priority to the development of an econometric model or models for use in the Intra-European air passenger markets.

A Working Group was given the following terms of reference:

- to give participating carriers an overview of existing models and methods, of the experience gained in their development and use, and of the feasibility of different approaches;
- to assess the general importance of various economic parameters in such forecasting models;
- to evaluate the availability and reliability of data sources for such parameters, historical as well as predicted values;
- to evaluate the possibility of establishing an exchange programme, or a joint data base, for the most important economic parameters;
- to evaluate the possibility of developing a model, or a firm methodology, for the present joint forecasting within AEA, e.g., "The AEA Short Term (5 year) forecast", or any other forecast which might be undertaken by the AEA.

Selection of Model Type

In early meetings, the AEA Working Group classified forecasting methods into the following types:

- Time series methods (e.g., Trend projection, exponential smoothing).
- Causal methods (e.g., Regression models, econometric models).
- Qualitative methods (e.g., Panel consensus, Delphi method).

Each AEA member was invited to describe relevant experience of each model type. All airlines reported use of Qualitative methods, frequently in conjunction with Time series or Causal models. General experience was that the time series models had worked well in the relatively stable economic environments prior to 1973, but, since then, they had been unable to cope with economic turbulence and rapid changes in airline fares. Those airlines with experience of causal models reported favourable results. The working group recognised that there would always be a place for entrepreneurial judgements but considered that the most appropriate basis should be some form of causal model.

In selecting the type of causal model, the group was guided by the experience of member airlines (especially in use of regression models) and by general reviews of the state of the art. The major considerations were:

- The accuracy of model type, especially at predicting trend changes.
- The cost and time required to construct a model, and
- The data requirements.

Highly expensive methods were ruled out, and regression analysis was selected for the AEA five year forecast as the cheapest suitable method. This model form is not necessarily best for all traffic forecasting problems. For very short term forecasting, trend projection may still be best; for very long term forecasting, input-output methods or technological forecasting techniques may be preferable. Where seasonal effects are important, Box-Jenkins may be applicable. But, for five year forecasts, where seasonal effects are not important, regression models are the most convenient way of using the available data to relate air travel to economic causes (Chambers 1970).

Selection of Variables

Initially, airlines sought those economic relationships which are most appropriate for European air traffic forecasting. For selected country-to-country traffic flows, actual traffic results were related to economic indicators (GNP, GDP, Trade, Industrial Production, etc.) and air fares. Additive and multiplicative models were explored using stepwise regression and, for almost all country pairs, the best results were obtained:

- using the combined GDP's of the two countries as the measure of economic growth (GDP's were weighted according to the proportion of tickets sold in each of the two countries);
- using average real airline revenue per passenger (in a constant currency) as the measure of the price of air travel (the deflator was the average of the national Consumer Price Indices, again weighting according to ticket sales);
- by a multiplicative relationship between combined GDP's and average real revenue per passenger.

The mathematical form of the model for a particular country pair is:

$$\text{Traffic} = K \times (f\ \text{GDP})^{b} \times (f\ \text{Average revenue per passenger})^{-c}$$

where:

$$f\text{GDP} = d \times \text{GDP}_1 + (1 - d) \times \text{GDP}_2$$

and

$$f \begin{pmatrix}\text{Average revenue} \\ \text{per passenger}\end{pmatrix} = \frac{\text{average revenue per passenger of both carriers}}{d \times \text{CPI}_1 + (1 - d) \times \text{CPI}_2}$$

GDP_1, GDP_2 are the respective real GDP indices of each country in the pair.

CPI_1, CPI_2 are the respective Consumer Price Indices of each country in the pair.

d, $(1 - d)$ are the proportions of ticket sales in each country.

K, b and c are model parameters for each country pair.

The parameter K is a constant for each flow.

The parameter b represents the tendency for traffic to increase as economic growth increases.

The parameter - c represents the tendency for traffic to decline as real fares increase.

Calibration of Models

For each traffic flow, regression analyses estimated the parameter values which gave the best fit between model estimates and the modified history of actual air traffic. (Actual traffic data were modified to allow for "special events" such as the French general strike etc., of May 1968 and the 1973 German ATC dispute.)

For some flows the models themselves were modified when the regression analyses gave a positive price elasticity, implying that traffic would increase as real fares increased. The positive price parameter was never statistically significant, but, because of the believed illogicality of the implied relationship, the fares variable was removed from the models for these flows. (Such apparently positive price elasticity may arise from either multi-collinearity of the independent variables or from the absence of data on other influencial factors such as surface transport costs.)

The calibrated models gave very good agreement with historic data with a few minor exceptions. These exceptions were traffic flows between neighbouring countries where air traffic is strongly influenced by scheduled frequencies and connections - where we could not expect economic relations alone to account for the traffic history. For completeness, and to allow for (unknown) future disruptions in overall traffic, all country-to-country models were nevertheless accepted as the basis for forecasting.

Model Interpretation

The model form is attractive because of the simplicity with which basic economic laws are represented. The parameters which form the exponents of the combined GDP's and the combined fares are interpretable respectively as the income elasticity and the price elasticity of demand.

But, there is an important qualification in the interpretation of the model parameters which precludes the use of the "price elasticity" parameter for tariff evaluations. The historic data shows significant correlation between GDP and real revenue per passenger. This correlation means that multiple regression cannot separate the contribution which each variable makes to traffic growth. So the "price elasticity" cannot be interpreted in isolation from "income elasticity". The "price elasticity" is not an independent measure of market response to tariff changes.

While correlation between the "independent" variables prevents multiple regression models being used for tariff purposes, the models are still valid for the main intention, that of forecasting traffic flows.

Model Value

Although the computing experiments demonstrated that relatively simple models could provide very high correlation between traffic trends and economic factors, it was still open to question whether the models are superior to informal methods, especially when the accuracies of the economic predictions are suspect.

While the only way of settling these issues is to make forecasts using the models and then await the outcome (see the section entitled "The Forecasting Process"), two reasonably strong arguments supported the use of models.

Firstly, previous AEA forecasts - established by discussion among experts in earlier years - were compared with the forecasts that the models would have produced if they had been used at that time. (Naturally, for a fair test, the models were calibrated omitting historic data for the period to be predicted.) Comparisons supported the models. In over 100 forecasts of traffic flows:

- the models were the more accurate in 80% of forecasts;

- the mean average deviations (average percentage error) of model forecasts were half that of the previous forecasts.

Therefore, viewed as tools to assist forecasting experts, the models were obviously of value.

Secondly, the inaccuracies of economic predictions appear to place an upper limit on the accuracy of traffic forecasts based on economic relationships. There would seem to be value in economic models only on condition that economic predictions are more accurate than informal traffic forecasts. But, since very high correlation exists between economic growth and air traffic growth, common sense suggests that this condition must hold; otherwise the many economic forecasters should use air traffic forecasts as the basis of the economic predictions. (To date, a practical test has not been made. Should one ever disprove "common sense", the results are unlikely to be published. But, if in future, European airline forecasters establish an International Bank, you may draw your own conclusions.)

Computer System Implementation

"APL is indispensable - if only I knew what for." - Jean Cocteau.

Interactive forecasting systems have been implemented in a number of time-sharing environments both by the AEA and by individual airlines. Depending upon the available software, approaches have ranged from straightforward use of self-contained statistical packages through ab initio programming of the complete system, including even the regression analyses functions. Thus the latest implementation, using SHARP APL, was influenced by experience of the comparative merits of alternatives. Additionally, forecasting system design was influenced by general requirements for all AEA data bases and systems.

Over the years European airlines have exchanged, through the AEA, statistics summarising the progress of airlines, both within Europe and worldwide, for passenger and freight markets.

Some examples are:

- monthly, seasonal, and annual traffic statistics by main IATA route groups;
- revenue and cost breakdowns - by route group and by aircraft type;
- point-to-point traffic (within Europe only);
- fleets, deployments, and utilisations;
- staff numbers and costs by airline function

For historic reasons, each exchange had separate files - sometimes on different computers - and analyses of relationships among various data items have been difficult. In transferring to the SHARP APL system, we wished to avoid the headaches of creating one integrated data base while still gaining the ability to relate all items to one another. These aims have been achieved:

(a) by creating separate data bases for each major data group, while

(b) developing compatible retrieval functions so that authorized users may simultaneously extract data from each separate data base.

Thus, for example, the point-to-point data can be related to the country-to-country traffic data used for forecasting or the national economic data can be compared with trends in traffic, revenues, and costs. Naturally, in obtaining the best of both worlds in terms of data base flexibility we have had to impose common standards for each individual data base. As far as the user is concerned, the data base designs cause no problem whatever. Admittedly, the programmers complained occasionally - but one accepts these comments philosophically - in the knowledge that, deep down, they really enjoy the challenge.

Further, as a general principle, all AEA data bases and systems are designed to be usable by member airlines who embrace a wide range of nationalities, cultures, and computing backgrounds. Great emphasis must be placed on ease and simplicity for use. Special requirements are:

(a) that the user interaction should look the same over all AEA data bases;

(b) common airline applications of existing packages (e.g., STATISTICS or MAGIC workspaces) should be accessible via the AEA standard interface;

(c) AEA data base designs and retrievals should be compatible with other Sharp aviation data bases and packages;

(d) users should be able to undertake individual analyses or experiments without changing the main AEA files.

The first two points ensure that unskilled users may still undertake standard tasks, while the third point ensures that more experienced users may relatively easily employ the full range of available data bases and programmes. The fourth point provides a general protection of AEA data integrity, by requiring that experimental results are written to special files.

Compared either with using existing APL functions alone or with writing functions tailored specifically for each application, the above requirements impose additional costs in return for user benefits. The main costs are in programme development and maintenance - and perhaps running costs - with some loss of flexibility in both data base maintenance and statistical analyses. The major gain, ease of use, minimises staff training problems, and increases both data and run reliability. While the benefits are less easily quantifiable than the costs, the trade-off seems justifiable because:

- European airlines have generally high standards for interactive systems. For in-house planning/market research systems, airlines have invested substantially in powerful on-line editors and command/control routines. (In particular, the "tailor-made" forecasting systems have much better interactive capability than that provided by the manufacturer's statistical packages, though with a loss of responsiveness and flexibility.)
- program development time, the major cost, was much less in APL than was experienced in compiler languages;
- interpretative languages (apart from BASIC) are not yet common in Europe and therefore one cannot depend upon user familiarity with APL.

AEA systems then, like Caesar's Gaul, can be divided into three parts:

(a) standard AEA data base retrieval and update functions and the user dialogue;

(b) special purpose functions, specific to each application, e.g., for forecasting, experimentation and reporting;

(c) standard Sharp system facilities such as the MAGIC data base language, statistical programmes, and public data bases.

In the forecasting application, most functions are either standard AEA routines or generally available facilities in the MAGIC or the STATISTICS workspaces. Unique functions are used only for particular calculations. The user communicates directly with AEA standard functions, and optionally, with MAGIC.

The main features of the AEA user interface are that each request

(a) has a standard format - a prompt followed by the current (default) value - normally the last user response to this request;

(b) rejects invalid responses immediately, and with appropriate error messages;

(c) allows for control responses HELP, STOP, etc., and also for advance responses to forthcoming requests;

(d) accepts a variety of input formats ranging from simple lists to complete APL expressions. In particular, all MAGIC selection formats and functions are valid.

These standards are automatically enforced by the simple expedient of having only one function which can read from the user terminal: all terminal I/O in all applications must therefore obey conventions of this single function. And, of course, in the forecasting system - whether the user intends to alter data, or perform regressions, or tabulate results - the terminal conventions are always identical.

The use of default values in terminal requests helps to achieve the maximum useful output for the minimum input. Data selections (e.g., of timeframe, or traffic flow) are transferred across the data maintenance/analysis/tabulation activities, so that the user may easily alter selected data and obtain tabulations of the revised forecasts. For experimental analyses, input is further reduced because modification of traffic or economic data leads automatically to selected model re-calibrations and forecast revisions.

As an overall assessment, the linking of AEA functions and general Sharp software provide methods for routine work which are at least as convenient as completely "tailor-made" programmes. The ease of incorporating extra MAGIC tabulations has proved very valuable but, as a personal opinion, it still seems not quite so easy to access the full power of the STATISTICS workspaces. We have not yet evolved a fully satisfactory means of making additional statistical analyses as easy as additional tabulations.

The Forecasting Process

"The ability to forsee that some things cannot be forseen is a very necessary quality" - Jean Jacques Rousseau.

Mr. C. Deetman of KLM has already described the relationships between forecasts of individual airlines, and those of industry bodies such as the AEA and IATA (Deetman 1975). The main purpose of the industry forecasts is to provide overall guidelines for airline corporate plans which help to avoid excessive capacity and competition ruinous to both airlines and customers. To be useful, therefore, the forecasts must be both reasonably accurate and acceptable. Cooperation among airline representatives is the essential process, with computer models being valuable in speeding up agreement of accurate forecasts.

In addition to divergent views on likely traffic trends, AEA members have divergent views on likely economic trends and airline commercial policies. Economic models which forecast traffic do not remove scope for debate; they change its form. As a framework for agreeing forecasts, airlines have devised a five-stage procedure:

(a) using the actual results (traffic and economic) of the previous year

- explain errors in previous forecasts;
- recalibrate the models;

(b) establish mutually consistent predictions of future trends in national economies (GDP, inflation rates, personal consumption) and airline revenue rates;

(c) using the models, forecast traffic levels both for country-to-country flows and for the whole of Europe;

(d) critically examine forecasts injecting influences not included in the models, e.g., airline commercial policies;

(e) agree traffic forecasts, documenting both the agreed forecasts and the model predictions with reasons for any differences.

Phases (a) and (b) take place at a joint meeting early each year. Traffic forecasts are agreed (phases (d) and (e)) at another joint meeting after airlines have studied the model results. Additionally, each year's forecasts may be updated in the autumn if unexpected economic trends appear.

In previous years, the computer models - although on time-sharing systems - were only used between meetings and inevitably, there were unplanned delays in resolving difficulties. This year, for the first time in Europe, the time-sharing system was used during the Forecasting Group Meeting itself. Thus the impact of alternative economic or revenue trends could be immediately assessed and airlines could reach agreement there and then. Partly through this advance, the forecasts were published some two months earlier than usual, a time saving which better fits the corporate planning cycle of most airlines.

In future, communications between airlines would improve, reducing delays even more, through member airlines connecting their in-house terminals to the same time-sharing system. Then data input and airline explorations of alternative futures would be substantially faster. Such progress would further enhance the ability of European airlines to collect, analyse, and interpret information and carry out joint or individual research.

Forecast Errors and Model Development

"Experience is the name men give to their mistakes." - Oscar Wilde.

Since model use began, two year's actual traffic (1976 and 1977) have been compared with previous forecasts, to ascertain the strengths and weaknesses in the models. For the very short term (2 years or less), forecast growth rates are applied to the latest results, whereas, further ahead the absolute traffic forecasts are regarded as the best guide. Although the overall model results have been encouraging, examination of detailed errors re-emphasises the necessity for judgement in calibrating and using models.

Figure 1

Comparisons of Forecasts and Actual Traffic (PAX NO. 000S)

	1976			1977		
	FORECAST	ACTUAL	% ERROR	FORECAST	ACTUAL	% ERROR
Forecasts made in 1976 (36 flows)	20884	21411	-2.5%	23199	22973	-1.0%
Forecasts made in 1977 (64 flows)	–	–	–	27747	27717	-0.1%

Actual 1976 traffic growth averaged 7.6%, compared with the forecast of 6.0%, an error of -1.6%. Interestingly, the subjective opinion had been that the model results were, if anything, too optimistic. The forecast report had stated that "in discussion of projections, there were opinions that some of the forecast results were approaching the upper limit of likely traffic growth" (i.e., diplomatically expressed scepticism). Yet, in the event, even the models underestimated the 1976 traffic upsurge which followed two years of stagnation.

In 1977, remarkable accuracy in the absolute traffic forecast arose from an overestimate of the growth rate. The major error in the 1977 growth rate forecast was almost entirely due to the U.K. traffic development which was severely disrupted by a prolonged ATC dispute.

Comparisons were made to distinguish errors due to:

- errors in estimates of national GDP growths or passenger revenue rates;
- model inaccuracy.

With the same models as the previous year, 1976 traffic was forecast again using the actual 1976 GDP's and revenues. Accuracy improved in only half the flows. For some of the other flows, random errors or errors due to the simple nature of the forecasting model itself outweighed errors in economic forecasts. But, in the main, errors in GDP estimates (mostly underestimates) were offset by countervailing errors in yield estimates.

Before jumping to the conclusion that we had the "ideal model" insofar as it produces accurate traffic forecasts even when the independent variables are wrong, we must recognise the element of "self-fulfilling prophecy" in short term air market forecasts. This effect may be especially strong for scheduled airlines because, by their nature, the production plan (i.e., the schedule) must be maintained while the "product" cannot be stored. For up to one year ahead, European airlines have already established pool schedules designed to satisfy estimated markets demands on each route. Of course, higher load factors, extra services and aircraft substitutions do permit some higher traffic when demand exceeds estimates. But there are limits on growth.

Conversely, when demand is below the estimates, sales promotion activities are strengthened - advertising, direct sales effort and promotion are stepped up; low-rate capacity is expanded. These measures reduce the average passenger yield but recover some of the loss in potential demand. Both the scheduled capacity constraints and the marketing efforts tend to damp forecasting errors in the short term. Over a longer period, however, we would still expect that errors in economic estimates will cause corresponding errors in traffic forecasts.

The good overall accuracies conceal some major discrepancies which were examined in detail and found to be due to special factors that could not reasonably be predicted. These were:

Traffic Flows	**Forecast Error due to**
1976 flows involving Spain (forecast too high)	political uncertainty following Franco's death.
1976 flows involving U.K. (forecast too low)	very rapid sterling devaluation.
1977 flows involving U.K. (forecast too high)	prolonged ATC dispute in U.K.
1977 flows involving Germany (forecast too high)	German GDP growth was half the prediction + political events in late 1977.
1977 flows Continent-Ireland (forecast too low)	Reaction to ATC dispute in U.K.

Judgements were needed about the long term consequences of those one-off events to decide whether or not the model calibrations should be based on the unusual results. Most 1976 Spanish flows were excluded from model calibrations as, effectively, were the Irish traffic results in 1977. But, for the U.K., the exceptionally high 1976 traffic upsurge was thought to offset the 1977 decline. Both year's results were included in model calibration. These decisions illustrate how the apparently objective process of calibrating models depends itself upon subjective judgements and why air traffic forecasting remains a mixture of art and science.

Future Research

Since the main errors have resulted neither from model inadequacies nor errors in independent variables, there is limited value in further model refinement. More valid models would not necessarily improve overall accuracy. But, for strategic planning, better understanding of market forces is sought and, in terms of model developments, the work programme for 1978 includes the following explorations:

(a) use of different independent variables for some flows (e.g., Consumer Expenditure);

(b) use of passenger kilometres rather than passenger numbers as the dependent variable (with corresponding changes to independent variables);

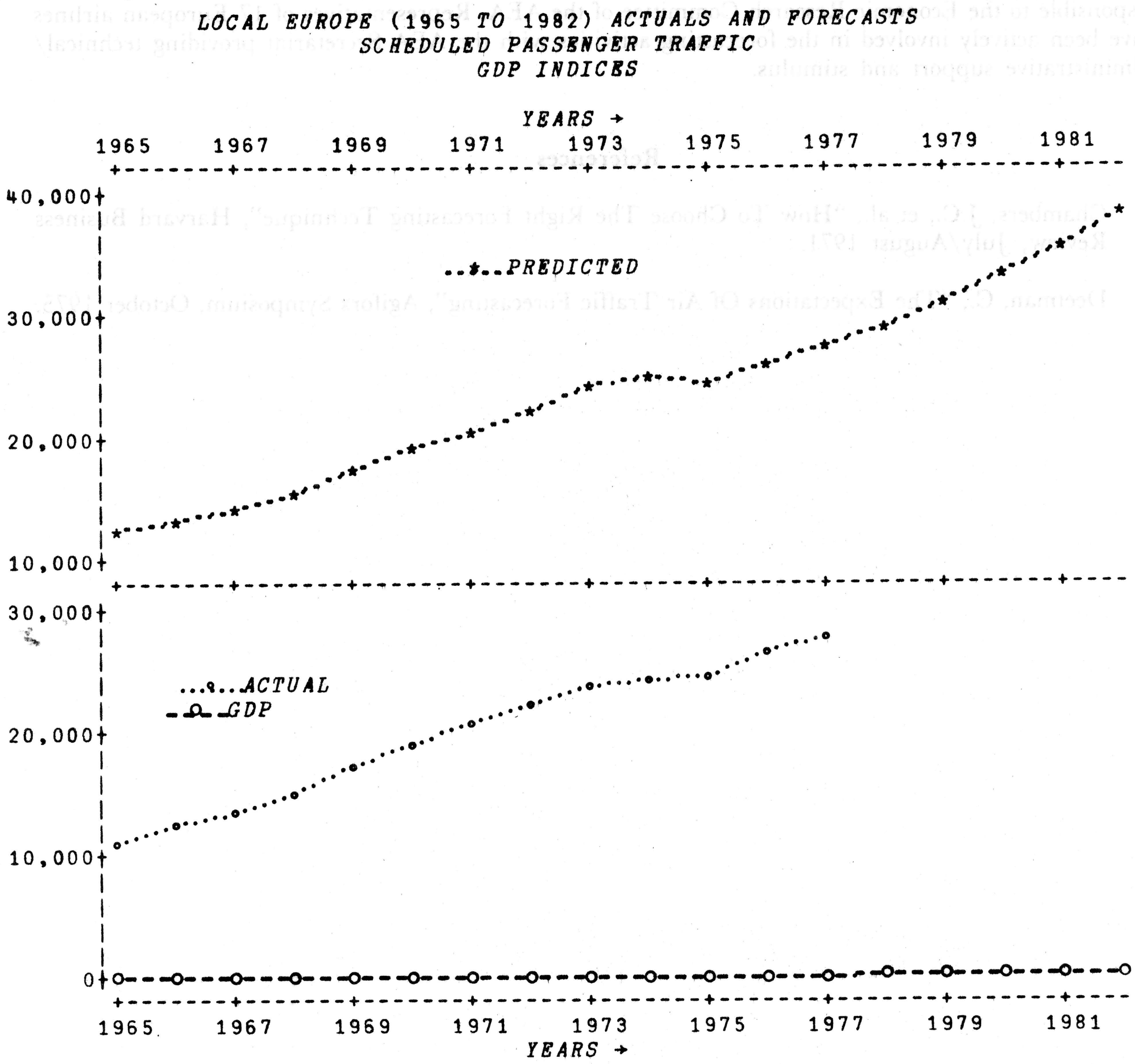
(c) impact of product changes on the small markets (e.g., relatively new flows, flows with major schedule changes);

(d) interaction between scheduled traffic and charter traffic;

(e) time trends in the model exponents.

In parallel, development continues on the corresponding statistical models which forecast European air freight and longer term trends of scheduled and charter traffic in major markets.

By the autumn, we expect to make some minor changes to the basic model form, but these will not greatly affect traffic forecasts. Studies (c) and (d) are expected to result in substantial model modifications for selected flows - those at a very early growth stage and those with substantial charter/scheduled traffic interaction. The final study, the examination of trend in the model parameters, will become more important in the longer term. Although European air markets are still far from saturation, before the end of the century we expect air traffic growth to approach more closely to general economic growth. Airlines are therefore keen to detect evidence of changing elasticities as soon as it appears. Figure 1 shows the most recent forecast for total international European traffic of AEA members. Similar forecasts exist for country-to-country flows, as well as for air freight and charter traffic.



Conclusion

As well as opening the door to faster, more accurate forecasting, the joint construction of statistical models represents an excellent example of the gains achievable through technical/commercial co-operation among European airlines. This latter aspect, itself of foremost importance to the AEA which furthers co-operation in all fields of air transport, will be strengthened by the creation of the first readily accessible data banks of European airline results. These provide AEA members with an important new foundation for research related to the development of air traffic and the economic aspects of European air transport in its widest sense. Such research will assist European airlines to anticipate and meet future economic challenges, thereby better meeting the travel needs of the markets we serve.

Acknowledgements

The above results arise from collaboration over the years by members of the AEA Forecasting Group, responsible to the Economic Research Committee of the AEA. Representatives of 17 European airlines have been actively involved in the forecasting activities with the AEA Secretariat providing technical/administrative support and stimulus.

References

1. Chambers, J.C., et al., "How To Choose The Right Forecasting Technique", Harvard Business Review, July/August 1971.

2. Deetman, C., "The Expectations Of Air Traffic Forecasting", Agifors Symposium, October 1975.

VARIATIONS ON THE PROJECTION FUNCTION AND THEIR APPLICATION TO THE ANALYSIS OF VARIANCE

Richard M. Heiberger
Wharton School
University of Pennsylvania
Philadelphia, Pa.

The familiar summation algorithm for the Analysis of Variance of the full factorial design can be expressed as a regression algorithm in which a vector of observations is projected into a space spanned by a set of patterned dummy variables. Investigation of the patterns leads to a more general recursive summation algorithm for all orthogonal designs including Latin Squares, two-way designs with proportional cell frequencies, blocking designs, fractional factorials, split plots, and repeated measures. The algorithm can be further extended to include all generally balanced designs, a class which also includes partially confounded effects and balanced and partially balanced incomplete block designs.

Keywords: Analysis of Variance Algorithm, Experimental Design, Generally Balanced Designs, Orthogonal Designs, Projection, Recursive Algorithms.

1. One-way Analysis of Variance (ANOVA)

The one-way Analysis of Variance (ANOVA) for data *Y* in groups with equal numbers of observations per group is usually computed from a two dimensional array *Y* by algorithm *A*1. (Function definitions are indicated in the text by *D* 'FUNCTION NAME'. The functions themselves are listed in the Appendix alphabetically.)

```
      D 'A1'

      D 'MEAN'
```

For sample data *Y1W*, with 3 groups and 4 observations per group,

```
      Y1W
 22  10  23  17
 17  15  11  19
  6  24  15  10
```

the sums of squares for groups, residual, and corrected are

```
      A1 Y1W
36.5  321.75  358.25
```

When the groups do not have the same number of observations an additional array *P* is needed to indicate the presence of data in the corresponding position of *Y*. The usual algorithm then becomes *A*2.

```
      D 'A2'

      P1WN
 1 1 1 1
 1 1 0 0
 1 1 1 0

      Y1WN
  22 10  23  17
  17 15   0   0
   6 24  15   0

      P1WN COMPRESS Y1WN
  22  10  23  17
  17  15
   6  24  15

      P1WN A2 Y1WN
16.222  270  286.22
```

2. Higher-way ANOVA

Many experimental designs classify the groups by more than one factor. With equal numbers of observations in all cells specified by the cross classification of the factors (thus excluding fractional designs), the algorithms analogous to *A*1 are easily defined. Most program libraries include a variant under the description of Analysis of Variance for the full factorial design with an arbitrary number of factors.

When the numbers of factors and levels per factor get large the cost of an experiment can rise rather quickly. To offset that increase a wide class of designs call for observations on only a balanced fraction of the possible cells. Algorithms for these designs can get complicated. Straightforward extensions of *A*1, for instance, run into bookkeeping problems for even the simplest of these designs, the Latin Square. With unequal numbers of observations in the cells extensions of *A*2 can lead to difficulties in interpretation.

The next section introduces a simple recursive algorithm appropriate for the class of orthogonal designs including the full factorial, fractional factorials such as the Latin Square, completely blocked designs, and many other orthogonal designs. The algorithm was originally defined by Wilkinson (1970). His paper includes the extensions of the algorithm to the class of generally balanced designs, a class which includes all designs in Cochran and Cox (1957), for example: balanced and partially balanced incomplete blocks, partially confounded designs, and certain types of non-orthogonal factors. The algorithm is implemented in GENSTAT (Nelder, et al, 1977).

3. ANOVA and Projection

The data array *Y* can be reshaped into a vector and the indicator array *P* restructured to a design matrix *X* with the function *XFP* (with inverse *PFX*). The usual ANOVA algorithm then becomes *A*3.

```
      D 'A3'

      D 'XFP'

      D 'PFX'

      X1WN
 1 0 0
 1 0 0
 1 0 0
 1 0 0
 0 1 0
 0 1 0
 0 0 1
 0 0 1
 0 0 1

      Y1WNV
22  10  23  17  17  15  6  24  15

      X1WN A3 Y1WNV
16.222  270  286.22
```

A set of equivalent expressions for *B* from `A3[2]` are shown in *EQB*.

```
      D 'EQB'
```

Line `EQB[7]` is recognized as the solution *B* that minimizes the expression:

```
      (Y-X+.×B)+.*2
```

for arbitrary *X* with linearly independent columns. Lines `EQB[6 8 9]` are equivalent for arbitrary *X*. For the special case of *X* consisting of orthogonal columns of ones and zeros, lines `EQB[3 4 5]` are also equivalent to `EQB[7]` and `EQB[3]` is computationally cheapest. Further exploration of the properties will be based on the equivalence of `EQB[3]` and `EQB[7]`. Computations will be based on `EQB[3]`.

The usual formulation for least squares regression using `EQB[7]` is given in *R*1.

```
      D 'R1'

      X1WN R1 Y1WNV
16.222  270  286.22
```

The only real difference between *A*3 and *R*1 is in lines *A*3[4] and *R*1[5]. Each element of *YHAT*, the projection of *Y* into the space spanned by the columns of *X*, is a repetition of an element of *B* selected by the non-zero column of the corresponding row of *X*.

```
      B
18  16  15

      YHAT
18  18  18  18  16  16  15  15  15
```

An equivalent indexing of *B* is provided by the *I* array, defined from *X* by the function *IFX* (with inverse *XFI*).

```
      D 'IFX'

      D 'XFI'

      I1WN
1  1  1  1  2  2  3  3  3
```

The array *I* can also be related directly to *P* by *IFP* and *PFI*.

```
      D 'IFP'

      D 'PFI'
```

When group membership is indicated by either *I* or *X* the restriction imposed by the implicit ordering of the two-way array *P*, that all observations in the same group appear in adjacent positions, is not required. This freedom will be used to advantage in the algorithm for higher-way designs.

A set of equivalent expressions for *YHAT* is given in *EQYHAT*.

```
      D 'EQYHAT'
```

The projection function *PRJ*, which constructs *YHAT* by projecting *Y* into the *X* space associated with the indices *I*, can now be defined by combining *EQB*[3] and *EQYHAT*[7].

```
      D 'PRJ'
```

One more equivalent algorithm *A*4 can be written using *PRJ*.

```
      D 'A4'

      I1WN A4 Y1WNV
16.222  270  286.22
```

The equivalence of *A4* to *A3* is seen from the reexpressions of the projection *A4*[3] in *EQP*, from the Pythagorean properties of the projection function in *EQPY*, and from the equivalent expressions for the sum of squares between groups in *EQSSB*.

```
      D 'EQP'

      D 'EQPY'

      D 'EQSSB'
```

The equivalences in *EQP* and *EQPY* are illustrated in Figure 1.

```
      FIGURE1

        /|Y         /|Y-YBAR
       / |         / |
      /  |        /  |
     /   |       /   |
    /    |      /    |
   /____⎕|     /____⎕|
YBAR   YHAT    0   YHAT-YBAR
```

4. Efficient Implementation

Part of the computational advantage of algorithms based on *EQB*[3] over those based on *EQB*[8 9] is their avoidance of the accumulation and inversion of the cross-product matrices. For designs with even moderately large numbers of factors or levels within factors the cross-product matrices can easily require more intermediate computer storage than is available. Algorithms based on *EQB*[7] require that *X* be stored in floating point rather than Boolean form. Algorithms based on *EQB*[3] have neither problem. Even the intermediate storage space needed to store *X* at each step can be avoided (at the cost of execution time) by respecifying the algorithm to conserve intermediate storage. A space efficient version of *A4* is given in *A5*.

```
      D 'A5'

      I1WN A5 Y1WNV
16.222  270  286.22
```

5. Algorithms for Higher-way ANOVA

Two-way ANOVA with equal numbers of observations per cell is computed for the three-way array *Y* by algorithm *A*6.

```
      D 'A6'
```

for sample data *Y2W*, with a 2 by 3 level crossed factorial structure and 4 observations per cell,

```
      Y2W
 4   6  53  68
 1  39   7  42
69  59  94  85

53  10  66  42
71  92  77  27
 5  74  33  64
```

the sums of squares for rows, columns, interactions, residual, and corrected total are

```
      A6 Y2W
315.38  2158.6  5990.3  12292  20756
```

Two-way ANOVA with unequal numbers of observations in the cells can cause interpretation difficulties. Orthogonal special cases do not. A general characterization of orthogonal designs will be given in Section 6.

One class of orthogonal two-way designs has proportional cell frequencies as defined by *PROPORTIONAL*.

```
      D 'PROPORTIONAL'
```

Equal cell frequencies are a special case. The changes required to get from *A*6 to *A*7 for proportional cell frequencies are similar to those needed to get from *A*1 to *A*2.

```
      P2WP
 1 1 0 0
 1 0 0 0
 1 0 0 0

 1 1 1 1
 1 1 0 0
 1 1 0 0
```

```
      PROPORTIONAL P2WP
1

      Y2WP
  4    6    0    0
  1    0    0    0
 69    0    0    0

 53   10   66   42
 71   92    0    0
  5   74    0    0

      P2WP COMPRESS Y2WP
  4    6
  1
 69

 53   10   66   42
 71   92
  5   74

      P2WP A7 Y2WP
2667  1472.8  4133.4  4321.7  12595

      D 'A7'
```

Another class of orthogonal designs is represented by the Latin Squares. These are three-way, main effects only designs which are formed as `N*-1` fractions of an `N*3` factorial. An example of a Latin Square is in `YLS` with treatment assignments specified in `TLS`.

```
      YLS
 38   50   19
 13   50   37
 38   33    4

      TLS
 1  2  3
 3  1  2
 2  3  1
```

The fraction is indicated by the array *PLS* computed by the function *PFT* (with inverse *TFP*).

```
      PLS
1 0 0
0 1 0
0 0 1

0 0 1
1 0 0
0 1 0

0 1 0
0 0 1
1 0 0

      YLSP
 38   0   0
  0  50   0
  0   0  19

  0   0  13
 50   0   0
  0  37   0

  0  38   0
  0   0  33
  4   0   0

      D 'PFT'

      D 'TFP'
```

The usual algorithm for the Latin Square is *A*8.

```
      D 'A8'
```

The result is sums of squares for rows, columns, treatments, residual, and corrected total

```
      TLS A8 YLS
188.67  900.67  602  364.67  2056
```

Although algorithms *A*6, *A*7, and *A*8 are obviously related they differ enough in detail that each can be used only for its specific design. Extension of this type of algorithm to a single program appropriate for all orthogonal designs would lead to expressions which appear very complex.

6. A Recursive Algorithm for Higher-way Orthogonal Designs

A simple extension of algorithm *A4* can be constructed to be used not only for all classes of designs considered so far, but also for the much larger class of orthogonal designs.

The one-way design was described to *A4* by a single index vector whose elements identify the group to which each observation belongs. That vector can be written as a one column matrix. An indicator that the ANOVA table consists of one effects' row indexed by the column is given by the Design array *D*.

```
      D1W←1 1 ρ1
```

The original vector can now be recovered by using the *BASED* function.

```
      I1WNM
 1
 1
 1
 1
 2
 2
 3
 3
 3

      I1WNM BASED D1W[1;]
1  1  1  1  2  2  3  3  3

      D 'BASED'
```

The two-way design must use a two column matrix of index vectors to describe the group classification of the observations. The rows of *I* specify the cells of the two-way design. The rows of the design array *D* associate the row, column, and interaction *X* spaces with appropriate columns of *I*. The indices used for each of the three projections are found by the *INDEX* function.

```
      I2WP
 1  1
 1  1
 1  2
 1  3
 2  1
 2  1
 2  1
 2  1
 2  2
 2  2
 2  3
 2  3

      D2W
1 0
0 1
1 1


      I2WP INDEX D2W
 1  1  1
 1  1  1
 1  2  2
 1  3  3
 2  1  4
 2  1  4
 2  1  4
 2  1  4
 2  2  5
 2  2  5
 2  3  6
 2  3  6

      D 'INDEX'
```

Orthogonal designs can now be defined. A set of *INDICES* are orthogonal if the projection of the *X* array associated with each column of the *INDICES* array onto the subspace orthogonal to the column of ones is orthogonal to the corresponding projection for every other column of *INDICES*. Otherwise they are confounded. The function *CONFOUNDED* detects confounding among the sets of *X* arrays associated with the columns of *INDICES*. Indices associated with interaction or nested effects will not be orthogonal to the indices of the associated main effects. Their structural relationship is determined by the design array *D* and made visible by the logical inner product *CONF* on the design array. The design is orthogonal if the only non-orthogonal indices are those determined by the design array *D*. Formally, orthogonality of the design can be determined by the *ORTHOGONAL* function.

```
      D 'ORTHOGONAL'

      D 'CONF'

      D 'CONFOUNDED'

      I2WP ORTHOGONAL D2W
1

      CONF D2W
 1 0 1
 0 1 1
 1 1 1

      CONFOUNDED I2WP INDEX D2W
 1 0 1
 0 1 1
 1 1 1
```

The analog of *A*4 appropriate for all orthogonal designs of arbitrary order is now displayed as *A*9.

```
      D 'A9'

      D1W A9 Y1WNV,I1WNM
16.222  270  286.22

      Y2WPV
4  6  1  69  53  10  66  42  71  92  5  74

      D2W A9 Y2WPV,I2WP
2667  1472.7  4133.4  4321.7  12595
```

The scalar *R* in *A*9 is the position number of the Residual Sum of Squares in the *SS* vector. Each row of the effects array *E* holds the projection of the *Y* vector into the subspace spanned by the columns of the *X* array associated with the corresponding row of *D* and orthogonal to the subspaces spanned by all preceding rows of *D*. The last two rows of *E* hold the Residual *Y* vector and the projection of *Y* into the subspace orthogonal to the mean.

The design and index arrays for the Latin Square in algorithm *A*8 are *DLS* and *ILS*.

```
       DLS
 1 0 0
 0 1 0
 0 0 1

       ILS
  1  1  1
  1  2  2
  1  3  3
  2  1  3
  2  2  1
  2  3  2
  3  1  2
  3  2  3
  3  3  1

       ILS ORTHOGONAL DLS
1

       YLSV
38  50  19  13  50  37  38  33  4

       DLS A9 YLSV,ILS
188.67  900.67  602  364.67  2056
```

7. Non-orthogonal Designs

In non-orthogonal, generally balanced designs the *X* arrays associated with each column of the *INDICES* array are not orthogonal. The values *E*[*K*;] in *A*9[7] are therefore off by a constant efficiency factor which represents the proportion of information confounded with earlier effects in the design. Wilkinson (1970) and Payne and Wilkinson (1977) discuss methods by which the efficiency factor is evaluated and the necessary adjustments made.

In non-orthogonal designs which do not exhibit general balance the efficiency factor is not constant for all components of each *E*[*K*;]. For these designs there is as yet no efficient summation algorithm. Extensions of the regression algorithm *R*1 are required. In these cases the sequence in which the rows of the *D* array should be ordered is not universally agreed upon.

8. Acknowledgments

The research leading to this paper was supported by the National Science Foundation through grant MCS-75-13994-A01.

Appendix

An alphabetical listing of function definitions indicated in the text by *D* 'FUNCTION NAME'.

```
       D 'A1'
     ∇ SS←A1 Y;B;SSB;SST;YBAR
[1]   ⍝FNS:MEAN
[2]    B←MEAN Y
[3]    YBAR←MEAN B
[4]    SSB←(⍴Y)[2]×(B-YBAR)+.*2
[5]    SST←(,Y-YBAR)+.*2
[6]    SS←SSB,(SST-SSB),SST
     ∇
```

```
       D 'A2'
     ∇ SS←P A2 Y;B;SSB;SST;YBAR
[1]    B←(+/Y)÷+/P
[2]    YBAR←(+/,Y)÷+/,P
[3]    SSB←(+/P)+.×(B-YBAR)*2
[4]    SST←(,Y-P×YBAR)+.*2
[5]    SS←SSB,(SST-SSB),SST
     ∇
```

```
       D 'A3'
     ∇ SS←X A3 Y;B;SSB;SST;YBAR
[1]   ⍝FNS:MEAN
[2]    B←(Y+.×X)÷+⌿X
[3]    YBAR←MEAN Y
[4]    SSB←(+⌿X)+.×(B-YBAR)*2
[5]    SST←(Y-YBAR)+.*2
[6]    SS←SSB,(SST-SSB),SST
     ∇
```

```
       D 'A4'
     ∇ SS←I A4 Y;E
[1]   ⍝FNS:MEAN;PRJ
[2]    E←(3,⍴Y)⍴Y-MEAN Y
[3]    E[1;]←I PRJ E[2;]
[4]    E[2;]←E[2;]-E[1;]
[5]    SS←E+.*2
     ∇
```

```
      D 'A5'
    ∇ SS←I A5 Y;B;K;N
[1]  ⍝FNS:MEAN
[2]   Y←Y-MEAN Y
[3]   SS←3ρY+.*2
[4]   B←(⌈/I)ρ0
[5]   N←(⌈/I)ρ0
[6]   K←1
[7]  L:B[I[K]]←B[I[K]]+Y[K]
[8]   N[I[K]]←N[I[K]]+1
[9]   →L×ι(ρY)≥K←K+1
[10]  SS[1]←N+.×(B÷N)*2
[11]  SS[2]←SS[3]-SS[1]
    ∇


      D 'A6'
    ∇ SS←A6 Y;N13;N23;N3;YBAR;Y1;Y2;Y12
[1]  ⍝FNS:MEAN
[2]   N23←×/(ρY)[2 3]
[3]   N13←×/(ρY)[1 3]
[4]   N3←(ρY)[3]
[5]   YBAR←MEAN,Y
[6]   Y1←(+/[2]+/[3] Y)÷N23
[7]   Y2←(+/[1]+/[3] Y)÷N13
[8]   Y12←(+/[3] Y)÷N3
[9]   SS←5ρ(,Y-YBAR)+.*2
[10]  SS[1]←N23×(Y1-YBAR)+.*2
[11]  SS[2]←N13×(Y2-YBAR)+.*2
[12]  SS[3]←N3×(,Y12-Y1∘.+Y2-YBAR)+.*2
[13]  SS[4]←SS[5]-+/SS[1 2 3]
    ∇


      D 'A7'
    ∇ SS←P A7 Y;N13;N23;N3;YBAR;Y1;Y2;Y12
[1]   N23←+/[2]+/[3]P
[2]   N13←+/[1]+/[3]P
[3]   N3←+/[3] P
[4]   YBAR←(+/,Y)÷+/,P
[5]   Y1←(+/[2]+/[3] Y)÷N23
[6]   Y2←(+/[1]+/[3] Y)÷N13
[7]   Y12←(+/[3] Y)÷N3
[8]   SS←5ρ(,Y-P×YBAR)+.*2
[9]   SS[1]←N23+.×(Y1-YBAR)*2
[10]  SS[2]←N13+.×(Y2-YBAR)*2
[11]  SS[3]←(,N3)+.×(,Y12-Y1∘.+Y2-YBAR)*2
[12]  SS[4]←SS[5]-+/SS[1 2 3]
    ∇
```

```
      D 'A8'
    ∇ SS←T A8 Y;N1;YBAR;YC;YR;YT
[1]   ⍝FNS:MEAN
[2]    N1←(ρY)[1]
[3]    YBAR←MEAN,Y
[4]    YR←MEAN Y
[5]    YC←MEAN⍉Y
[6]    YT←MEAN(ρY)ρ(,Y)[⍋,T]
[7]    SS←5ρ(,Y-YBAR)+.*2
[8]    SS[1]←N1×(YR-YBAR)+.*2
[9]    SS[2]←N1×(YC-YBAR)+.*2
[10]   SS[3]←N1×(YT-YBAR)+.*2
[11]   SS[4]←SS[5]-+/SS[1 2 3]
    ∇
```

```
      D 'A9'
    ∇ SS←D A9 YI;E;INDICES;K;R;Y
[1]   ⍝FNS:BASED;MEAN;PRJ
[2]    Y←YI[;1]
[3]    INDICES← 0 1 ↓YI
[4]    R←1+(ρD)[1]
[5]    E←((R+1),ρY)ρY-MEAN Y
[6]    K←1
[7]   L:E[K;]←(INDICES BASED D[K;]) PRJ E[R;]
[8]    E[R;]←E[R;]-E[K;]
[9]    →L×⍳R>K←K+1
[10]   SS←E+.*2
    ∇
```

```
      D 'BASED'
    ∇ I←INDICES BASED D
[1]    I←⍉D/INDICES
[2]    I←1+(⌈/I)⊥I-1
    ∇
```

```
      D 'COMPRESS'
    ∇ Z←P COMPRESS Y;RZ
[1]    Z←⍕Y
[2]    RZ←ρZ
[3]    Z←RZρ(,P)⍀(,P)⌿((×/ρY),(¯1↑RZ÷ρY))ρZ
    ∇
```

```
      D 'CONF'
    ∇ Z←CONF D
[1]    Z←D∨.∧⍉D
    ∇
```

```
        D 'CONFOUNDED'
      ∇ Z←CONFOUNDED INDICES;N;I;J;XI;XJ
[1]    ⍝FNS:MEAN;XFI
[2]     N←(⍴INDICES)[2]
[3]     Z←(N,N)⍴1
[4]     →0×⍳N=1
[5]     I←1
[6]    LI:J←I+1
[7]     XI←XFI INDICES[;I]
[8]     XI←XI-(⍴XI)⍴MEAN⍉XI
[9]    LJ:XJ←XFI INDICES[;J]
[10]    XJ←XJ-(⍴XJ)⍴MEAN⍉XJ
[11]    Z[J;I]←Z[I;J]←⎕CT∨.≤,|(⍉XI)+.×XJ
[12]    →LJ×⍳N≥J←J+1
[13]    →LI×⍳N>I←I+1
      ∇
```

```
        D 'EQB'
      ∇ EQB
[1]    ⍝IN:B;X;Y
[2]     B
[3]     (Y+.×X)÷+⌿X
[4]     Y+.×X÷(⍴X)⍴+⌿X
[5]     (⍉X÷(⍴X)⍴+⌿X)+.×Y
[6]     (⌹X)+.×Y
[7]     Y⌹X
[8]     (⌹(⍉X)+.×X)+.×(⍉X)+.×Y
[9]     ((⍉X)+.×Y)⌹(⍉X)+.×X
      ∇
```

```
        D 'EQP'
      ∇ EQP
[1]    ⍝IN:E;I;Y;YBAR;YHAT
[2]    ⍝FNS:MEAN;PRJ
[3]     E[1;]
[4]     I PRJ(Y-MEAN Y)
[5]     (I PRJ Y)-(I PRJ MEAN Y)
[6]     (I PRJ Y)-MEAN Y
[7]     YHAT-MEAN Y
[8]     YHAT-YBAR
      ∇
      D 'EQPY'
    ∇ EQPY
[1]  ⍝IN:E;Y;YBAR;YHAT
[2]   E[3;]+.*2
[3]   (Y-YBAR)+.*2
[4]   ((Y-YHAT)+.*2)+((YHAT-YBAR)+.*2)
[5]   (E[2;]+.*2)+(E[1;]+.*2)
    ∇
```

```
      D 'EQSSB'
    ∇ EQSSB
[1]  ⍝IN:B;I;X;Y;YBAR;YHAT
[2]  ⍝FNS:IFX:MEAN:PRJ
[3]   (+⌿X)+.×(B-YBAR)*2
[4]   ((X+.×B)-YBAR)+.*2
[5]   (YHAT-YBAR)+.*2
[6]   (B[IFX X]-YBAR)+.*2
[7]   (B[I]-YBAR)+.*2
[8]   (I PRJ(Y-MEAN Y))+.*2
    ∇
```

```
      D 'EQYHAT'
    ∇ EQYHAT
[1]  ⍝IN:B;I;X;YHAT
[2]  ⍝FNS:IFX
[3]   YHAT
[4]   X+.×B
[5]   B[X+.×⍳(⍴X)[2]]
[6]   B[IFX X]
[7]   B[I]
    ∇
```

```
      D 'IFP'
    ∇ I←IFP P
[1]   I←(,P)/,P×⍉(⌽⍴P)⍴⍳(⍴P)[1]
    ∇
```

```
      D 'IFX'
    ∇ I←IFX X
[1]   I←X+.×⍳(⍴X)[2]
    ∇
```

```
      D 'INDEX'
    ∇ Z←I INDEX D;K;N
[1]  ⍝FNS:BASED
[2]   N←(⍴D)[1]
[3]   Z←((⍴I)[1],N)⍴0
[4]   K←1
[5]  L:Z[;K]←I BASED D[K;]
[6]   →L×⍳N≥K←K+1
    ∇
```

```
      D 'MEAN'
    ∇ Z←MEAN Y
[1]   Z←(+/Y)÷¯1↑⍴Y
    ∇
```

```
      D 'ORTHOGONAL'
    ∇ Z←I ORTHOGONAL D
[1]  ⍝FNS:CONF;CONFOUNDED;INDEX
[2]   Z←∧/,(CONF D)=CONFOUNDED I INDEX D
    ∇
```

```
      D 'PFI'
    ∇ P←PFI I
[1]  ⍝FNS:PFX;XFI
[2]   P←PFX XFI I
    ∇
```

```
      D 'PFT'
    ∇ P←PFT T;I;J
[1]  ⍝FNS:BASED
[2]   P←(×/3⍴⍴T)⍴0
[3]   J←(⍴T)⍴⍳1↑⍴T
[4]   I←⍉J
[5]   P[((,I),(,J),[1.1],T) BASED 1 1 1]←1
[6]   P←(3⍴⍴T)⍴P
    ∇
```

```
      D 'PFX'
    ∇ P←PFX X;F
[1]   F←+⌿X
[2]   P←F∘.≥⍳⌈/F
    ∇
```

```
       D 'PRJ'
     ∇ YHAT←I PRJ Y;B;X
[1]   ⍝FNS:XFI
[2]    X←XFI I
[3]    B←(Y+.×X)÷+⌿X
[4]    YHAT←B[I]
     ∇


       D 'PROPORTIONAL'
     ∇ Z←PROPORTIONAL P;N
[1]    N←+/P
[2]    Z←∧/,N=(+/N)∘.×(+⌿N)÷+/,N
     ∇


       D 'R1'
     ∇ SS←X R1 Y;B;SSB;SST;YBAR;YHAT
[1]   ⍝FNS:MEAN
[2]    B←Y⌹X
[3]    YHAT←X+.×B
[4]    YBAR←MEAN Y
[5]    SSB←(YHAT-YBAR)+.*2
[6]    SST←(Y-YBAR)+.*2
[7]    SS←SSB,(SST-SSB),SST
     ∇


       D 'TFP'
     ∇ T←TFP P
[1]    T←P+.×⍳1↑⍴P
     ∇


       D 'XFI'
     ∇ X←XFI I
[1]    X←I∘.=⍳⌈/I
     ∇


       D 'XFP'
     ∇ X←XFP P;RXT
[1]    RXT←(⍴P)×1,(⍴P)[1]
[2]    X←⍉RXT⍴P,RXT⍴0
[3]    X←(,P)⌿X
     ∇
```

References

1. Cochran, William G., and Gertrude M. Cox (1957). **Experimental Designs**, Second Edition. Wiley, New York.

2. Nelder, J. A., et al (1977). **GENSTAT User's Manual**. Rothamsted Experimental Station, Harpenden, Herts., England.

3. Payne, Roger W., and Graham N. Wilkinson (1977). "A General Algorithm for Analysis of Variance", **Applied Statistics, Journal of the Royal Statistical Society (Series C)**, 26, No. 3, pp. 251-260.

4. Wilkinson, Graham N. (1970). "A General Recursive Algorithm for Analysis of Variance," **Biometrika, 57**, pp. 19-46.

APL STANDARDS AND DOCUMENTATION WITHIN AN EVOLUTIONARY ENVIRONMENT

Alan Harrison
British Nuclear Fuels Limited
Seascale, Cumbria, England

Abstract

The paper describes documentation and standards that have been found useful at BNFL, Windscale, together with techniques for coping with what was essentially an evolutionary approach to the development of production systems in APL. Two main topics are explored:

(i) standards appropriate to an evolving technology (especially in comparison to more traditional areas of computing).

(ii) the problem of documentation and standards for "on-line" interactive production systems which must be flexible and quickly extensible.

Introduction

This paper discusses various aspects of APL standards and documentation found useful within British Nuclear Fuels Limited (BNFL).

Initially, I define the term "standard" in the context of this paper and then briefly describe the history of APL within BNFL, Windscale and discuss the term "evolutionary environment" as used within the paper. Following this is an examination of the major constraints to standards and documentation for commercial applications. Finally, various aspects of standards and documentation are listed.

A Definition of "Standards"

The word "standard" in the English language has many meanings ranging from "a flag" to "an upright support". Within the computer industry the definition has been refined to imply strict rules which must be adhered to in the production of computer systems. This enforced conformity is intended to avoid confusion and aid the production of clear maintainable systems.

For the purpose of this paper I have considered "standards" more generally, incorporating useful techniques for system design and the type of person capable of utilising them.

BNFL's Use of APL

BNFL, Windscale has been using APL since the initial venture of I.P. Sharp Associates into the United Kingdom market. Originally APL was used as a problem-solving tool by the Operational Research and Computing Services groups. In the early years APL proved its worth as an effective tool for small scale use by relatively highly trained specialists. However, it was never seriously considered as an effective language for producing commercial systems. The major reason for not taking this initiative was the popular "high-cost" myth generated by the "sceptic-APL" press.

In early 1976 a new group was formed, at Windscale, to provide a support function to the production management. The data this new department required was available on our "in-house" computers but it was relatively inaccessible because they needed an extremely flexible information system which could be quickly extended. This was necessary because at the time it was very difficult to define the information requirements in more detail. Additionally, as outlined below, we had to adopt a totally different style of system design to satisfy fully the requirements of flexibility, speed of implementation and extensibility.

So we embarked on our first commercial APL system. Initially, brief discussions were held with the department concerned and from these meetings a basic system was produced which attempted to satisfy what I perceived as their immediate requirements. Throughout this period I attempted to keep the system within an interactive framework of consultation, modification and improvement. In the past two years, therefore, the system has evolved into a large and comprehensive information system which is data independent and has consequently been used as an interrogatory tool for other applications. The immediate and continuing successful use of the system as a management aid has fully justified the decision to adopt an evolutionary approach to the design.

The original intention was to allow the system to evolve until it became stable. At this point a conversion to one of the more conventional languages (e.g. COBOL) would be considered. The conversion of the system has been rejected because the cost is prohibitive when compared to the expected savings in running costs.

Within this period, various other systems, adopting the same techniques of evolutionary design, have been implemented as production systems. These systems cover such areas as graphics, planning and modelling. At the same time, various systems were implemented at BNFL head office in the area of financial planning and support of commercial negotiations.

At the present time BNFL are at the crossroads of APL policy encountered by all companies using the language. Should we be using APL as a tool for highly trained specialists alone or, alternatively, expanding its use to that of a full commercial language? If we proceed in the latter direction, how will it affect our staffing and organization? Is APL an economically viable option when compared to our "in-house" resources? If so, should we go "in-house" with APL or stay with commercial bureaux? etc. etc.

The Evolutionary Environment

APL as implemented on computers at this time does not satisfy Iverson's original notation nor does it cover all aspects of data structures used today. Additionally, the interfaces between APL and the computer are still deficient, although advances are being made. So, APL as a computer system is in a process of evolution.

The computer industry has been slow in recognising the capabilities of APL, probably because of the fundamental effect on the structure and personnel of the typical data-processing department if APL began to replace COBOL, or its counterparts. So, the computer industry is evolving to accommodate APL.

At a company level, the attitude and reaction of BNFL to APL is evolving, while, as an individual, I am still modifying and expanding my ideas on how APL should be used.

These four areas of change demonstrate the evolutionary environment in which APL system design is performed. Within a relatively static environment, such as FORTRAN or COBOL, stringent standards can be applied. It is my suggestion that this type of standard cannot be applied to a dynamic environment, as described above, independently of whether APL standards should be laid down as strict rules.

How then can this affect the design of commercial systems? Within the system design and implementation of "conventional" programming language systems a great deal of time is spent in analysing and defining what is required. This is necessary because to change systems after implementation has begun is usually complicated, costly and time consuming. I feel this approach has two major drawbacks on more than a trivial system:

a) When the system has been completed it is possible that the requirements have changed since the design phase and the system is not as effective as originally planned.

b) Once the system has been designed there is an inevitable gap during the programming phase when the liason between computer staff and customer is at a low level, leaving the customer to cope with his as yet unsolved problems. I feel that this is not conducive to the maintenance of the customer relationship, especially when the computer team return and the customers' department is suddenly inundated with training, parallel running, and all the problems with personnel that occur during this period.

APL offers a realistic alternative to this approach because it can be written and modified quickly and efficiently. By adopting an evolutionary approach to systems design these problems can be minimised. The number of people involved can be reduced, ideally to one or two - the person with the problem, and the person solving it. Systems can start small and grow in response to patterns of use and changing requirements, offering advantages to both designer and customer as they work together towards the solution. The designer can produce a basic system very quickly and this can evolve towards an optimal solution, especially in terms of cost. Meanwhile the user has something to use and gain experience with, while refining his definition of his system's requirements. The typical arguments concerning ambiguities and omissions in the specification do not arise, because the specifications evolve out of the use of the basic system. Through working with the user, the designer becomes more aware of his customers' problems, and both benefit from the good working relationships generated by this approach.

The techniques discussed below are those that I have found useful using this method of evolutionary design.

Constraints in Using APL for Commercial Applications

In general, there are three major constraints upon a system designer. The relative importance of each of these constraints dictates not only the level of success and subsequent use of his system but also the techniques he can utilise in its production.

The primary consideration is obviously cost. The system must be designed and implemented to a budget figure and subsequent running costs must be acceptably low to ensure that the system is adopted. Both these constraints affect standards and documentation. Standards adopted by the designer should be such as to enhance maintainability, thus reducing running costs without increasing design and implementation costs significantly. In general, costs tend to encourage the minimisation of standards and documentation.

A second consideration is time. The time spent developing and maintaining systems should be kept to a minimum, which has a limiting affect on the type and quantity of standards and documentation available to the designer.

The final constraint that has a significant impact upon standards and documentation is maintainability; a successful system should be maintainable with the minimum of cost and time. This encourages the designer to adopt more stringent and detailed standards than he would otherwise consider.

The environment within which the designer works and the specific application on which he is working will necessitate a compromise on these three constraints. The standard data processing technique is to adopt a rigorous set of rules and apply them to every job. I believe APL gives the designer the ability to choose the optimal compromise, and as such, his standards are a set of techniques from which he can pick the ones most relevant to his present problem. APL standards should be guidelines to a good "style" rather than strict laws which must be obeyed. If we adopt strict rules we run the risk of restraining the designer from making the most effective use of the language.

Obtaining and Imposing Standards

In general, when attempting to adopt standards, the natural tendency is to look at the industry for guidance. In this area the suppliers of APL have, in the past, been lacking, although the problem is now achieving more consideration. I think this is due to the early "desk-calculator" uses of APL and the way it has been marketed. The realisation that APL is an effective and economical commercial data processing language has made standards and documentation more important.

The majority of standards obtained, so far, from the industry have been implied; that is, they are those adhered to by the vendors of APL, for example, the use of on-line documentation to describe the purpose of objects within a workspace. I intend to discuss these techniques and others I have found useful in generating clear maintainable code, rather than programming standards concerned with the comparable efficiency of various APL operators. This is because these techniques are the ones which I believe are relevant to the use of APL as a commercial "data processing" language rather then its use as a problem-solving tool.

The most important "standard" in the use of APL for commercial systems design is undoubtedly the standard of person used to design and implement such systems. To make this statement is extremely simple - to define the type of person is much more difficult. The most appropriate character is not necessarily the type of person involved in current commercial systems design. In fact the "APL-er" is more likely not to be recruited from the typical personalities involved in commercial computer programming, although I have met many good APL designers with traditional computing backgrounds.

To work in an evolutionary design environment, the ideal person should have a pleasant, easily acceptable character to enable him to work closely and continuously with his customer, and the ability to be inquisitive without appearing to pry, and diplomatic without seeming sly. To gain the confidence of his customer and generate a relationship of trust he should be patient, self assured, logical and perceptive. In short he should possess to a high degree those personal qualities common to the successful practitioner of systems analysis and O.R.

Obviously, he must be proficient in APL, but ideally he should have an empathy with the language in that he naturally structures his problems in an APL-like manner.

If the choice of this person is correct he will be capable of choosing the specific techniques necessary to produce a clear maintainable system for each application he tackles.

This may be the factor which will limit the successful application of APL.

Useful Techniques

This section details various techniques I have found useful in writing APL systems. A few of these techniques are implementation dependent - these cases relate to the I.P. Sharp Associates' APL system. The order in which the standards are discussed is not intended to imply any specific ranking.

1. "On-line" User Documentation

The vendors of APL imply a standard in this area by documenting the majority of public libraries with *DESCRIBE* functions, etc., in the workspace. I can see two advantages for the bureaux.

a) All information relating to the system is readily available from the terminal. This overcomes a significant number of communication problems caused by the diversity of users' geographical locations and eases the updating of both system and documentation.

b) There is no cost penality incurred by the supplier, although this cannot be said for the user.

This is a very nice solution to the documentation of evolving systems. However, it is expensive in monetary and workspace area terms. (The latter problem can be overcome, at the expense of the former, by utilising the file subsystem to store this information.)

I use "on-line" documentation to generate a brief description of the system, with a reference to an operating manual and a support contact, so that the user is provided with immediate assistance, and given the route to obtain more detailed information, at minimal cost.

2. "On-line" Program Documentation

This is the "on-line" information found beneficial to the subsequent maintenance of the system. Some areas of specific interest are:

- A short comment at the beginning of each function indicating the parameters, results, date of last update, i.e. sufficient information to enable, ideally, its reconstruction.
- Additional comments wherever necessary.
- The use of *⎕LX* to provide at least a title to the system. (See section 5 below.)
- The assignment of meaningful names to objects. This is especially important with globals - as the name is the only "on-line" documentation available.
- The reservation of an area at the beginning of each file for its documentation.

3. Workspace Documentation Aids

Various tools are provided by the majority of computer bureaux to aid the documentation of workspaces, e.g. *SEARCH* and *EDIT* functions in library 777 on the I.P. Sharp system. Although of possible use in the final documentation, I find the cost prohibitive in an evolving environment. So my normal practice is **not** to use them.

4. Project File

The basic concept behind the "project file" is to ensure that documentation evolves as the system develops, rather than waiting until the system has been implemented.

This file would typically contain:

- The original objectives and any subsequent modification.
- All communications concerning the system.
- System configuration.
- Standards adopted.
- User manual (See section 5).
- How to introduce new users into a multiuser system.
- Security controls.
- File documentation.
- Workspace listings.
- An explanation of globals.
- Special terminal characteristics.
- Restart conventions - what the restart system can recover from and how this is achieved.
- Possible improvements and enhancements.
- Why specific techniques have been chosen.
- Any useful debugging or programming techniques.
- Coding "funnies".

5. User Manual

This is the document supplied to every user of the system as his prime source of information under normal circumstances. Again, the typical contents will be:

- A contact for when the manual is insufficient.
- How to get into the system (including sign-on procedures).
- System description.
- System objectives.
- Description of input options and formats.
- Description of output options and formats.
- Default states.
- Error messages.
- Possible problems (i.e., line noise, etc.) and how to cope with them.
- Plenty of examples.

6. System Design

a) The basic concept within a system design is the choice of the most appropriate structure. Most problems are best expressed in a modular fashion; APL with its functional structure naturally lends itself to the expression of this type of problem. It is important to make each function a logical unit and not an artificial breakdown of the problem. The problem is successively broken down into smaller logical units until it is clearly comprehensible. Our experience has shown that, generally, functions over 10-15 lines in length are not sufficiently understandable.

b) The development of "PACKAGES" on SHARP APL has resulted in significant improvements to the clarity of multi-file systems because all variables, functions and documentation relating to a specific data file may, conveniently, be stored on that file. The workspace contains only general code with reference to the specific file functions.

c) We have found it useful to use `⎕LX` to autostart systems to perform operations such as: printing a title, tying files, initialisation, or passing control to a system executive.

d) In general we have found it useful to provide simple forms of restartability to commercial workspaces.

e) In the implementation of large systems we have preferred to use PACKAGES for overlaying functions rather than the technique of linking workspaces.

7. Programming Techniques

Our experience has shown that the following techniques are useful in providing clear understandable functions:

- Use meaningful names (especially for global variables and functions).
- Localise variables wherever possible.
- Consider converting global variables to functions, thus aiding documentation and preventing accidental modification.
- Adopt a convention for labels which makes them easily identifiable as such, e.g., `L1`, `L2` etc.
- Use only one labelled exit from a function, e.g., `L99`.
- Do not over complicate lines by increasing their length unnecessarily.
- Avoid artificial constructs, e.g., `X←5,0 ρ Y←99`.

Acknowledgements

The views expressed are the personal views of the author and I would like to acknowledge my thanks to BNFL, Windscale for providing the opportunity to give this paper and to the individuals, notably R.H. Peebles and P.L. O'Neill, instrumental in the development of APL at Windscale. To I.P. Sharp Associates I would like to express my gratitude for inviting me to give this paper and providing the incentive to write it.

Finally, I would like to acknowledge the valuable contribution made by Joe Friswell (BNFL Windscale, O.R. Department). I am indebted to him for our fruitful discussions and his helpful suggestions on both the content and structure of this paper.

PROGRAMMING STYLE IN APL

Kenneth E. Iverson
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

When all the techniques of program management and programming practice have been applied, there remain vast differences in quality of code produced by different programmers. These differences turn not so much upon the use of specific tricks or techniques as upon a general manner of expression, which, by analogy with natural language, we will refer to as **style**. This paper addresses the question of developing good programming style in APL.

Because it does not rest upon specific techniques, good style cannot be taught in a direct manner, but it can be fostered by the acquisition of certain habits of thought. The following sections should therefore be read more as examples of general habits to be identified and fostered, than as specific prescriptions of good technique.

In programming, as in the use of natural languages, questions of style depend upon the purpose of the writing. In the present paper, emphasis is placed upon clarity of expression rather than upon efficiency in space and time in execution. However, clarity is often a major contributor to efficiency, either directly, in providing a fuller understanding of the problem and leading to the choice of a better, more flexible, and more easily documented solution, or indirectly, by providing a clear and complete model which may then be adapted (perhaps by programmers other than the original designer) to the characteristics of any particular implementation of APL.

All examples are expressed in `0`-origin. Examples chosen from fields unfamiliar to any reader should perhaps be skimmed lightly on first reading.

1. Assimilation of Primitives and Phrases

Knowledge of the bare definition of a primitive can permit its use in situations where its applicability is clearly recognizable. Effective use, however, must rest upon a more intimate knowledge, a feeling of familiarity, an ability to view it from different vantage points, and an ability to recognize similar uses in seemingly dissimilar applications.

One technique for developing intimate knowledge of a primitive or a phrase is to create at least one clear and general example of its use, an example which can be retained as a graphic picture of its behavior when attempting to apply it in complex situations. We will now give examples of creating such pictures for three important cases, the outer product, the inner product, and the dyadic transpose.

Outer product. The formal definition of the result of the expression `R←A∘.fB` for a specified primitive f and arrays `A` and `B` of ranks `3` and `4` respectively, may be expressed as:

```
R[H;I;J;K;L;M;N]↔A[H;I;J] f B[K;L;M;N]
```

Although this definition is essentially complete, it may not be very helpful to the beginner in forming a manageable picture of the outer product.

To this end it might be better to begin with the examples:

```
      N∘.+N←1 2 3 4                                N∘.×N
 2 3 4 5                                        1  2  3  4
 3 4 5 6                                        2  4  6  8
 4 5 6 7                                        3  6  9 12
 5 6 7 8                                        4  8 12 16
```

and emphasize the fact that these outer products are the familiar addition and multiplication tables, and that, more generally, `A∘.fB` yields a **function table** for the function f applied to the sets of arguments *A* and *B*.

One might reinforce the idea by examples in which the outer product illuminates the definition, properties, or applicability of the functions involved. For example, the expressions `S∘.×S←¯3 ¯2 ¯1 0 1 2 3`, and `×S∘.×S` yield an interesting picture of the rule of signs in multiplication, and the expressions `R∘.=V` and `R∘.≤V` and `' ⋆'[R∘.=V]` (with `V←(X-3)×(X←1+⍳7)-5` and with *R* specified as the range of *V*, that is, `R←8 7 6 5 4 3 2 1 0 ¯1`) illustrate the applicability of outer products in defining and producing graphs and bar charts. These and other uses of outer products as function tables are treated in Iverson [1].

Useful pictures of outer products of higher rank may also be formed. For example, `D∘.∨D∘.∨D←0 1` gives a rank three function table for the **or** function with three arguments, and if *A* is a matrix of altitudes of points in a rectangular area of land and *C* is a vector of contour levels to be indicated on a map of the area, then the expression `C∘.≤A` relates the points to the contour levels and `+⌿C∘.≤A` gives the index of the contour level appropriate to each point.

Inner Product. Although the inner product is perhaps most used with at least one argument of rank two or more, a picture of its behavior and wide applicability is perhaps best obtained (in the manner employed in Chapter 13 of Reference 1) by first exploring its significance when applied to vector arguments. For example:

```
      P←2  3  5  7 11
      Q←2  0  2  1  0
```

```
      +/P×Q
21
```

Total cost in terms of price and quantity.

```
      ⌊/P+Q
3
```

Minimum trip of two legs with distances to and from connecting point given by *P* and *Q*.

```
      ×/P⋆Q
700
```

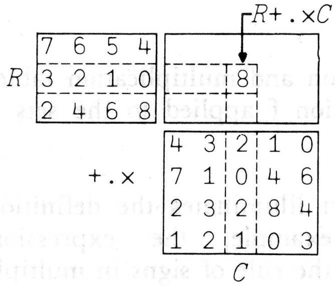
The number whose prime factorization is specified by the exponents *Q*.

```
      +/P×Q
21
```

Torque due to weights *Q* placed at positions *P* relative to the axis.

The first and last examples above illustrate the fact that the same expression may be given different interpretations in different fields of application.

The inner product is defined in terms of expressions of the form used above. Thus, *P*+.×*Q* ↔ +/*P*×*Q* and, more generally for any pair of scalar functions f and g, *P*f.g*Q* ↔ f/*P*g*Q*. The extension to arrays of higher rank is made in terms of the definition for vectors; each element of the result is the inner product of a pair of vectors from the two arguments. For the case of matrix arguments, this can be represented by the following picture:



The +.× inner product applied to two vectors *V* and *W* (as in *V*+.×*W*) can be construed as a **weighted sum** of the vector *V*, whose elements are each "weighted" by multiplication by the corresponding elements of *W*, and then summed. This notion can be extended to give a useful interpretation of the expression *M*+.×*W*, for a matrix *M*, as a weighted sum of the column vectors of *M*. Thus:

```
      W←3 1 4
      ⎕←M←1+3 3⍴⍳9
 1 2 3
 4 5 6
 7 8 9
      M+.×W
17 41 65
```

This result can be seen to be equivalent to writing the elements of *W* below the columns of *M*, multiplying each column vector of *M* by the element below it, and adding.

If *W* is replaced by a boolean vector *B* (whose elements are zeros or ones), then *M*+.×*B* can still be construed as a weighted sum, but can also be construed as **sums over subsets** of the rows of *M*, the subsets being chosen by the 1's in the boolean vector. For example:

```
      B←1 0 1
      M+.×B
4 10 6
      B/M
 1 3
 4 6
 7 9
      +/B/M
4 10 16
```

Finally, by using an expression of the form *M*×.⋆*B* instead of *M*+.×*B*, a boolean vector can be used to apply multiplication over a specified subset of each of the rows of *M*. Thus:

```
      M×.⋆B
3 24 63
      ×/B/M
3 24 63
```

This use of boolean vectors to apply functions over specified subsets of arrays will be pursued further in the section on generalization, using boolean matrices as well as vectors.

Dyadic transpose. Although the transposition of a matrix is easy to picture (as an interchange of rows and columns), the dyadic transpose of an array of higher rank is not, as may be seen by trying to compare the following arrays:

```
      A                   2 1 3⍉A              3 2 1⍉A
ABCD                ABCD                AM
EFGH                MNOP                EQ
IJKL                                    IU
                    EFGH
MNOP                QRST                BN
QRST                                    FR
UVWX                IJKL                JV
                    UVWX
                                        CO
                                        GS
                                        KW

                                        DP
                                        HT
                                        LX
```

The difficulty increases when we permit left arguments with repeated elements which produce "diagonal sections" of the original array. This general transpose is, however, a very useful function and worth considerable effort to assimilate. The following example of its use may help.

The associativity of a function f is normally expressed by the identity:

```
      Xf(YfZ)↔(XfY)f
Z
```

and a **test** of the associativity of the function on some specified domain `D←1 2 3` can be made by comparing the two function tables `D∘.f(D∘.fD)` and `(D∘.f D)∘.fD` corresponding to the left and right sides of the identity. For example:

```
      D←1 2 3
      ⎕←L←D∘.-(D∘.-D)            ⎕←R←(D∘.-D)∘.-D         L=R
 1  2  3                  ¯1 ¯2 ¯3                0 0 0
 0  1  2                  ¯2 ¯3 ¯4                0 0 0
¯1  0  1                  ¯3 ¯4 ¯5                0 0 0

 2  3  4                   0 ¯1 ¯2                0 0 0
 1  2  3                  ¯1 ¯2 ¯3                0 0 0
 0  1  2                  ¯2 ¯3 ¯4                0 0 0

 3  4  5                   1  0 ¯1                0 0 0
 2  3  4                   0 ¯1 ¯2                0 0 0
 1  2  3                  ¯1 ¯2 ¯3                0 0 0
```

```
      ∧/,L=R
0
      ⎕←L←D∘.+(D∘.+D)            ⎕←R←(D∘.+D)∘.+D          L=R
 3 4 5                       3 4 5                      1 1 1
 4 5 6                       4 5 6                      1 1 1
 5 6 7                       5 6 7                      1 1 1

 4 5 6                       4 5 6                      1 1 1
 5 6 7                       5 6 7                      1 1 1
 6 7 8                       6 7 8                      1 1 1

 5 6 7                       5 6 7                      1 1 1
 6 7 8                       6 7 8                      1 1 1
 7 8 9                       7 8 9                      1 1 1
```

For the case of logical functions, the test made by comparing the function tables can be made complete, since the functions are defined on a finite domain `D←0 1`. For example:

```
      D←0 1
      ∧/,(D∘.∨(D∘.∨D))=((D∘.∨D)∘.∨D)
1
      ∧/,(D∘.≠(D∘.≠D))=((D∘.≠D)∘.≠D)
1
      ∧/,(D∘.⍲(D∘.⍲D))=((D∘.⍲D)∘.⍲D)
0
```

Turning to the identity for the distribution of one function over another we have expressions such as:

```
      X×(Y+Z)↔(X×Y)+(X×Z)
```

and

```
      X∧(Y∨Z)↔(X∧Y)∨(X∧Z)
```

Attempting to write the function table comparison for the latter case as:

```
      L←D∘.∧(D∘.∨D)
      R←(D∘.∧D)∘.∨(D∘.∧D)
```

we encounter a difficulty since the two sides *L* and *R* do not even agree in rank, being of ranks `3` and `4`.

The difficulty clearly arises from the fact that the axes of the left and right function tables must agree according to the names in the original identity; in particular, the *X* in position `0` on the left and in positions `0` and `2` on the right implies that axes `0` and `2` on the right must be "run together" to form a single axis in position `0`. The complete disposition of the four axes on the right can be seen to be described by the vector `0 1 0 2`, showing where in the result each of the original axes is to appear. This is a paraphrase of the definition of the dyadic transpose, and we can therefore compare *L* with `0 1 0 2⍉R`. Thus:

```
      ∧/,(D∘.∧(D∘.∨D))=0 1 0 2⍉((D∘.∧D)∘.∨(D∘.∧D))
1
```

The idea of thorough assimilation discussed thus far in terms of primitive expressions can be applied equally to commonly used phrases and defined functions. For example:

⍳⍴*V*	The indices of vector *V*
⍳⍴⍴*A*	The axes of *A*
×/⍴*A*	The number of elements in *A*
V[⍋*V*]	Sorting the vector *V*
M[⍋+⌿*R*<.-⍉*R*←*M*,0;]	Sorting the rows of *M* into lexical order
⍉*F*⍉*M*	Applying to columns a function *F* defined on rows

Collections of commonly used phrases and functions may be found in Perlis and Rugaber [2] and in Macklin [3].

2. Function Definition

A complex system should best be designed not as a single monolithic function, but as a structure built from component functions which are meaningful in themselves and which may in turn be realized from simpler components. In order to interact with other elements of a system, and therefore serve as a "building block", a component must possess inputs and outputs. A defined function with an explicit argument, or arguments, and an explicit result provides such a component.

If a component function produces **side effects** by setting global variables used by other components, the interaction between components becomes much more difficult to analyze and comprehend than if communication between components is limited to their explicit arguments and explicit results. Ideally, systems should be designed with communication so constrained and, in practice, the number of global variables employed should be severely limited.

Because the fundamental definition form in APL (produced by the use of ∇ or by ⎕*FX*, and commonly called the **del** form) is necessarily general, it permits the definition of functions which produce side effects, which have no explicit arguments, and which have no explicit results. The **direct** form which uses the symbols ⍺ and ⍵ (as defined in Iverson [4]) exercises a discipline more appropriate to good design, allowing only the definition of functions with explicit results, and localizing all names which are specified within the function, thereby eliminating side effects outside of it.

The direct form of definition may be either simple or conditional. The latter form will be discussed in section 6. The simple form may be illustrated as follows. The expression

```
F:⍵+4÷⍺
```

may be read as "*F* is defined by the expression ⍵+4÷⍺, where ⍺ represents the first argument of *F* and ⍵ represents the second". Thus 8 *F* 3 yields 3.5.

If a direct definition is to produce a machine executable function, the definition must be translated by a suitable function. For example, if this translation is called *DEF*, then:

```
      DEF
F:α+÷ω
F
      3 F 4
3.25

      DEF
P:+/α×ω*⍳⍴α
P
      1 3 3 1 P 4
125
```

```
      DEF
SORT:ω[⍋ω]
SORT
      SORT 3 1 4 3 6 2 7 6
1 2 3 3 4 6 6 7

      DEF
POL:(ω∘.*⍳⍴α)+.×α
POL
      1 3 3 1 POL 0 1 2 3 4
1 8 27 64 125
```

The direct form of definition will be used in the examples which follow. The question of the translation function *DEF* is discussed in Appendix A.

3. Generality

It is often possible to take a function defined for a specific purpose and modify it so that it applies to a wider class of problems. For example, the function `AV:(+/ω)÷⍴ω` may be applied to a numeric vector to produce its average. However, it fails to apply to average all rows in a matrix; the simple modification `AV2:(+/ω)÷¯1↑⍴ω` not only permits this, but applies to average the vectors along the last axis of any array, including the case of a vector.

The problem might also be generalized to a weighted average, in which a vector left argument specifies the weights to be applied in summation, the result being normalized by division by the total weight. Again this function could be defined to apply to a vector right argument in the form `WAV:(+/α×ω)÷+/α`, but, applying the inner product in the manner discussed in the preceding section, we may define a function which applies to matrices:

```
      WAV2:(ω+.×α)÷+/α
```

Thus:

```
      ⎕←M←3 4⍴⍳12
 0  1  2  3
 4  5  6  7
 8  9 10 11
      W←2 1 3 4
      W WAV2 M
1.9 5.9 9.9
```

The same function may be interpreted in different ways in different disciplines. For example, if column *I* of *M* gives the coordinates of a mass of weight `W[I]`, then `W WAV2 M` is the center of gravity of the set of masses. Moreover, if the elements of *W* are required to be non-negative, then the result `W WAV2 M` is always a point in the convex space defined by the points of *M*, that is a point within the body whose vertices are given by *M*. This can be more easily seen in the following equivalent function:

```
      WAV3:⍵+.×(⍵÷+/⍺)
```

in which the weights are normalized to sum to 1.

Striving to write functions in a general way not only leads to functions with wider applicability, but often provides greater insight into the problem. We will attempt to illustrate this in three areas, functions on subsets, indexing, and polynomials.

Functions on subsets. It is often necessary to apply some function (such as addition or maximum) over all elements in some subset of a given list. For example, to sum all non-negative elements in the list `X←3 ¯4 2 0 ¯3 7`, we might first define the boolean vector which identifies the desired subset, then select the set, and then sum it:

```
      X
3 ¯4 2 0 ¯3 7
      X≥0                     (X≥0)/X              +/(X≥0)/X
1 0 1 1 0 1               3 2 0 7              12
```

In general, if *B* is a boolean vector which defines a subset, we may write `+/B/X`. However, as seen in the discussion of inner product, this may also be written in the form `X+.×B`, and in this form it applies more generally to a boolean matrix (or higher rank array) in which the columns (or vectors along the leading axis) determine the different subsets. For example, if

```
      ⎕←B←(4⍴2)⊤⍳2*4
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
```

then the columns of *B* represent all possible subsets of a vector of four elements, and if `X←2 3 5 7` then:

```
      X+.×B
0 7 5 12 3 10 8 15 2 9 7 14 5 12 10 17
```

yields the sums over all subsets of *X*, including the empty set (`0 0 0 0`), and the complete set (`1 1 1 1`).

It is also easy to establish that

```
      X×.*B
1 7 5 35 3 21 15 105 2 14 10 70 6 42 30 210
```

yields the products over all subsets, and that (for non-negative vectors *X*) the expression

```
      X⌈.×B
0 7 5 7 3 7 5 7 2 7 5 7 3 7 5 7
```

yields the maxima over all subsets of *X*. This last expression holds only for non-negative values of *X*, but could be replaced by the more general expression `M+(X-M←⌊/X)⌈.×B`. A more general approach to this problem (in terms of a new operator) is discussed in Section 2 of Iverson [5].

If we have a list *A* with repeated elements, and if we need to evaluate some costly function *F* on each element of *A*, then it may be efficient to evaluate *F* only on the **nub** of *A* (consisting of the distinct elements of *A*) and then distribute the results to the appropriate positions to yield *FA*. Thus:

Function	**Definition**
Nub	`NUB:((⍳⍴⍵)=⍵⍳⍵)/⍵`
Distribution	`DIS:(NUB)∘.=⍵`
Example	`F:⍵*2`

Example

```
      A←3 2 3 5 2 3
      NUB A
3 2 5
      DIS A
1 0 1 0 0 1
0 1 0 0 1 0
0 0 0 1 0 0
      F A
9 4 9 25 4 9
      F NUB A
9 4 25
      (F NUB A)+.×DIS A
9 4 9 25 4 9
```

From the foregoing it may be seen that an inner product post-multiplication by the distribution matrix `DIS A` distributes the results `F NUB A` appropriately. The distribution function may also be used to perform aggregation or summarization. For example, if *C* is a vector of costs associated with the account numbers recorded in *A*, then summarization of the costs for each account may be obtained by pre-multiplication by `DIS A`. Thus:

```
      C←1 2 3 4 5 6
      (DIS A)+.×C
10 7 4
```

Indexing. If *M* is a matrix of `N←¯1↑⍴M` columns, and if *I* and *J* are scalars, then element `M[I;J]` can be selected from the ravel `R←,M` by the expression `R[(N×I)+J]`. More generally, if *K* is a two-rowed matrix whose columns are indices to elements of *M*, then these elements may be selected much more easily from *R* (by the expression `R[(N×K[0;])+K[1;]]`) than from *M* itself. Moreover, the indexing expression can be simplified to `R[(N,1)+.×K]`, or to `R[(⍴M)⊥K]`.

The last form is interesting in that it applies to an array *M* of any rank *P*, provided that *K* has *P* rows. More generally, it applies to an index array *K* of any rank (provided that (ρρ*M*)=1↑ρ*K*) to produce a result of shape 1↓ρ*K*. To summarize, we may define a general indexing function:

```
SUB:(,α)[(ρα)⊥ω]
```

and use it as in the following examples:

```
      ⎕←M←3 3ρι9                           ⎕←K←3|2 5ρι10
 0 1 2                                 0 1 2 0 1
 3 4 5                                 2 0 1 2 0
 6 7 8
      M SUB K
2 3 7 2 3
      M SUB 3|2 3 5ρι30
 0 4 8 0 4
 8 0 4 8 0
 4 8 0 4 8
      (4 4 4ρι4*3) SUB 4|3 2 6ρ2×ι36
  0 42  0 42  0 42
  0 42  0 42  0 42
```

This use of the base value function in the expression (ρα)⊥ω correctly suggests the possible use of the inverse expression (ρα)⊤ω to obtain the indices to an array α in terms of the index to its ravel (that is, ω).

Polynomials. If *F*:+/α×ω*ιρα, then the expression *C F X* evaluates the polynomial with coefficients *C* for the scalar argument *X*. The more general function:

```
P:(ω∘.*ιρα)+.×α
```

applies to a vector right argument and (since ω∘.*ιρα is then a matrix *M*, and since *M*+.×α is a linear function of α) emphasizes the fact that the polynomial is a linear function of its coefficients. If (ρω)=ρα, then *M* is square, and if the elements of ω are all distinct (that is, (ρω)=ρ*NUB*)), then *MIN* is non-singular, and the function:

```
FIT:(⌹ω∘.*ιρα)+.×α
```

is inverse to *P* in the sense that:

C ↔ (*C P X*) *FIT X*   and   *Y* ↔ (*Y FIT X*)*P X*

In other words, if *Y*←*F X* for some scalar function *F*, then *Y FIT X* yields the coefficients of the polynomial which fits the function *F* at the points (arguments) *X*. For example:

```
      3⍕Y←*X←0 .5 1 1.5
1.000 1.649 2.718 4.482
      3⍕C←Y FIT X
1.000 1.059  .296  .364
      3⍕C P X
1.000 1.649 2.718 4.482
```

The function *F* can be defined in a neater equivalent form, using the dyadic form of ⌹, as `FIT:α⌹ω∘.*⍳⍴α`. Moreover, the more general function:

```
      LSF:α⌹ω∘.*⍳N
```

(which depends upon the global variable *N*) yields the *N* coefficients of the polynomial of order `N-1` which best fits the function `α←Fω` in the least squares sense. Thus:

```
      N←4
      3⍕Y LSF X
1.000 1.059  .296  .364
      N←3                                     N←2
      3⍕C←Y LSF X                             3⍕C←YLSF X
1.014  .631 1.115                       .735 2.303
      3⍕C P X                                 3⍕C P X
1.014 1.608 2.759 4.468                 .735 1.886 3.038 4.189
```

The case `N←2` yields the best straight line fit. It can be used, for example, in estimating the "compound interest" or "growth rate" of a function that is assumed to be approximately exponential. This is done by fitting the logarithm of the values and then taking the exponential of the result. For example:

```
      X←0 1 2 3 4 5
      3⍕Y←300×1.09*X
 300.000 327.000 356.430 388.509 423.474 461.587
      N←2
      3⍕E←(⍟Y) LSF X
 5.704  .086
      *E
300 1.09
      3⍕(*E[0])×(*E[1])*X
 300.000 327.000 356.430 388.509 423.474 461.587
      3⍕Y←Y+?6⍴⎕RL←50
 300.000 355.000 395.430 434.509 454.474 508.587
      3⍕E←(⍟Y) LSF X
 5.749  .099
      *E
313.7594974 1.104368699
      3⍕(*E[0])×(*E[1])*X
 313.759 346.506 382.671 422.609 466.717 515.427
```

The growth rate is `*E[1]`, and the estimated compound interest rate is therefore given by the function

```
      ECI:100×¯1+*1↓(⍟α) LSF ω
```

For example:

```
      1⍕I←Y ECI X
10.4
      3⍕(*E[0])×(1+.01×I)*X
313.759 346.506 382.671 422.609 466.717 515.427
```

General considerations can often lead to simple solutions of specific problems. Consider, for example, the definition of a "times" function *T* for the multiplication of polynomials, that is:

```
      (C P X)×(D P X) ↔ (C T D) P X
```

The function *T* is easily shown to be linear in both its left and right arguments, and can therefore be expressed in the form `C+.×B+.×D`. The array *B* is a boolean array whose unit elements serve to multiply together appropriate elements of *C* and *D*, and whose zeros suppress contributions from other pairs of elements. The elements of *B* are determined by the exponents associated with *C*, with *D*, and with the result vector, that is, `⍳⍴C` and `⍳⍴D` and `⍳⍴1↓C,D`. For each element of the result, the "deficiency" of each element of the exponents associated with *D* is given by the table `S←(⍳⍴1↓C,D)∘.-⍳⍴D`, and the array *B* is obtained by comparing this deficiency with the contributions from the exponents associated with *C*, that is, `(⍳⍴C)∘.=S`. To summarize, the times function may be defined as follows:

```
      T:α+.×(αBω)+.×ω
      B:(⍳⍴α)∘.=(⍳⍴1↓α,ω)∘.-⍳⍴ω
```

For example:

```
      ⎕←E←(C←1 2 1) T (D←1 3 3 1)
1 5 10 10 5 1
```

Since the expression `α+.×(αBω)` yields a matrix, it appears that the inverse problem of defining a function *DB* (divided by) for polynomial division might be solved by inverting this matrix. To this end we define a related function *BQ* expressed in terms of *E* and *C*, rather than in terms of *C* and *D*:

```
      BQ:(⍳⍴α)∘.=(⍳⍴ω)∘.-⍳1+(⍴ω)-⍴α
```

and consider the matrix `M←C+.×C BQ E`.

The expression `(⌹M)+.×E` fails to work properly because *M* is not square, and we recognize two cases, the first being given by inverting the top part of *M* (that is, `⌹(2⍴⌊/⍴M)↑M`) and yielding a quotient with high-order remainder, and the second by inverting the bottom part and yielding a quotient with low order remainder. Thus:

```
      DBHO:(D↑α)⌹(2⍴D← ⌊/⍴M)↑M←ω+.×ωBQα
      DBLO:(D↑α)⌹(2⍴D←-⌊/⍴M)↑M←ω+.×ωBQα
```

For example:

```
      E←1 5 10 10 7 4
      C←1 2 1
      ⎕←Q←E DBHO C
1 3 3 1
      0⍕R←E-C T Q
  0   0   0   0   2   3
```

```
      E←47101051
      C←1 2 1
      ⎕←Q←E DBLO C
1 3 3 1
      0⍕R←E-C T Q
 3 2 0 0 0 0
```

The treatment of polynomials is a prolific source of examples of the insights provided by precise general functions for various processes, insights which often lead to better ways of carrying out commonly-needed hand calculations. For example, a function `E` for the expansion of a polynomial `C` (defined more precisely by the relation `(E C)P X ↔ C P X+1`) can be defined as:

```
      E:(BC ρω)+.×ω                                     BC:(ιω)∘.!ιω
```

Working out an example shows that manual expansion of `C` can be carried out be jotting down the table of binomial coefficients of order `ρC` (that is, `BC ρω`) and then taking a weighted sum of its columns, the weights being the elements of `C`.

4. Identities

An identity is an equivalence between two different expressions. Although identities are commonly thought of only as tools of mathematical analysis, they can be an important practical tool for simplfying and otherwise modifying expressions used in defining functions.

Consider, for example, a function `F` which applied to a boolean vector suppresses all `1`'s after the first. It could be used, for example, in the expression `(~F X='D')/X` to suppress the first `D` in a character string `X`. The function could be defined as `F:(ωι1)=ιρω`. However, the following identity holds:

```
      (ωι1)=ιρω↔<\ω
```

and we may therefore use one or other of the equivalent functions:

```
      F:(ωι1)=ιρω                                       G:<\ω
```

One may react to a putative identity in several ways: accept it on faith and use it as a practical tool, work some examples to gain confidence and a feeling for why it works, or prove its validity in a general way. The last two take more time, but often lead to further insights and further identities. Thus the application of the functions `F` and `G` to a few examples might lead one to see that `G` applies in a straightforward way to the rows of a matrix, but `F` does not, that both can be applied to locate the first **zero** by the expressions `~F~B` and `~G~B`, and (perhaps) that the latter case (that is, `~<\~B`) can be replaced by the simpler expression `≤\B`.

As a second example, consider the expression `Y←((~B)/X),B/X` with `B←X≤2`. The result is to **classify** the elements of `X` by placing all those in a specified class (those less than or equal to 2) at the tail end of `Y`. More generally, we may define a **classification** function `C` which classifies the elements of its right argument according to its boolean left argument:

```
      C:((~α)/ω),α/ω
```

For example:

```
      X←3 1 4 7 2
      ⎕←B←X≤2
0 1 0 0 1
      B C X
3 4 7 1 2
```

Since the result of `C` is a permutation of its right argument, it should be possible to define an equivalent function in the form `ω[V]`, where `V` is some permutation vector. It can be shown that the appropriate permutation vector is simply `⍋α`. For example:

```
      ⍋B                                         X[⍋B]
 0 2 3 1 4                                  3 4 7 1 2
```

Thus:

```
      P:ω[⍋α]    and    C:((~α)/ω),α/ω
```

are equivalent functions.

For any given function there are often related functions (such as an inverse) of practical interest. For example, if `V←B C X`, then there is some inverse function `CI` such that `B CI V` yields `X`. Moreover, the definition of a related function may be much easier to derive from one of several different equivalent definitions of the original function than from the others. Thus the definition of the inverse `CI` may not be immediately evident from the definition `C`, but from the definition `P` it is clear that what is needed is the inverse permutation. Thus:

```
      CI:ω[⍋⍋α]

      ⎕←V←B C X                                  B CI V
3 4 7 1 2                                   3 1 4 7 2
```

Finally, a given formulation of a function may suggest a simple formulation for a similar function. For example, the application of the function `P` with a left argument containing a single `1` can be seen to effect a rotation of that suffix of the right argument marked off by the location of the `1`. This suggests the following formulation for a function which rotates each of the segments marked off by the `1`'s in the left argument:

```
      RS:ω[⍋α++\α]

      1 0 0 1 0 0 0 1 0 RS 'ABCDEFGHI'
BCAEFGDIH
```

Dualities. We will now consider one class of very useful identities in some detail. The most familiar example of the class is known as deMorgan's law and is expressed as follows:

```
      X∧Y ↔ ~(~X)∨(~Y)
```

Useful related forms of deMorgan's law are:

```
      ∧/V ↔ ~∨/~V
      ∧\V ↔ ~∨\~V
      M∨.∧N ↔ (~M)∧.∨(~N)
```

DeMorgan's law concerns a relation between the functions **and, or,** and **not** (`∧ ∨ ~`), and we say that `∧` is the **dual of** `∨` **with respect to** `~`. Each of the boolean functions of two arguments possess a dual with respect to `~`. For example, `X≤Y ↔ ~(~X)<(~Y)`, and from this the three related identities `≤/V ↔ ~</~V`, etc.) follow in the manner shown above. The five dual pairs of boolean functions are:

```
∨  ⍱  <  =  >
∧  ⍲  ≤  ≠  ≥
```

These dualities are frequently useful in simplifying expressions used in logical selections. For example, we have already seen the use of the duality between `≤` and `<` to replace the expression `~<\~ω` by `≤\ω`.

Useful dualities are not limited to boolean functions. For example, **maximum** and **minimum** (`⌈` and `⌊`) are dual with respect to **arithmetic** negation (`-`) as follows:

```
X⌈Y ↔ -(-X)⌊(-Y)
```

Again the related forms of duality follow.

More generally, duality is defined in terms of any monadic function *M* and its inverse *MI* as follows: a function *F* is said to be the dual of a function *G* with respect to *M* if:

```
X F Y ↔ MI (M X)G(M Y)
```

In the preceding examples of duality, each of the monadic functions used (`~` and `-`) happened to be self-inverse and *MI* was therefore indistinguishable from *M*.

The general form includes the duality with respect to the natural logarithm function `⍟` which lies at the root of the use of logarithm tables and addition to perform multiplication, namely:

```
×/X ↔ *+/⍟X
```

The use of base ten logarithms rests similarly on duality with respect to the monadic function `10⍟ω` and its inverse `10*ω`:

```
×/X ↔ 10*+/10⍟X
```

5. Proofs

A proof is a demonstration of the validity of an identity based upon other identities or facts already proven or accepted. For example, deMorgan's law may be proved by simply evaluating the two supposedly equivalent expressions (`X∧Y` and `~(~X)∨(~Y)`) for all possible combinations of boolean values of *X* and *Y*:

X	Y	X∧Y	~X	~Y	(~X)∨(~Y)	~(~X)∨(~Y)
0	0	0	1	1	1	0
0	1	0	1	0	1	0
1	0	0	0	1	1	0
1	1	1	0	0	0	1

An identity which is useful and important enough to be used in the proofs of other identities is commonly called a theorem. Thus:

Theorem 1 `(A×B)∘.×(P×Q) ↔ (A∘.×P)×(B∘.×Q)`

We will prove theorem 1 itself for vectors `A,B,P`, and `Q` by calling the results of the left and right expressions `L` and `R` and showing that for any indices `I` and `J`, the values of `L[I;J]` and `R[I;J]` agree. We do this by writing a sequence of equivalent expressions, citing at the right of each expression the basis for believing it to be equivalent to the preceding one. Thus:

```
L[I;J]
((A×B)∘.×(P×Q))[I;J]                 Def of L
(A×B)[I]×(P×Q)[J]                    Def of ∘.×
(A[I]×B[I])×(P[J]×Q[J])              Def of vector ×

R[I;J]
((A∘.×P)×(B∘.×Q))[I;J]               Def of R
(A∘.×P)[I;J]×(B∘.×Q)[I;J]            Def of matrix ×
(A[I]×P[J])×(B[I]×Q[J])              Def of ∘.×
(A[I]×B[I])×(P[J]×Q[J])              × associates and commutes
```

Comparison of the expressions ending the two sequences completes the proof.

We will now state a second theorem (whose proof for vector variables is given in Iverson [6]), and use it in a proof that the product of two polynomials `C P X` and `D P X` is equivalent to the expression `+/,(C∘.×D)×X*(⍳⍴C)∘.+⍳⍴D`:

Theorem 2 `+/,V∘.×W ↔ (+/V)×(+/W)`

Thus:

Theorem 3

```
(C P X)×(D P X)
(+/C×X*E←⍳⍴C)×(+/D×X*F←⍳⍴D)          Def of P
+/,(C×X*E)∘.×(D×X*F)                 Theorem 2
+/,(C∘.×D)×((X*E)∘.×(X*F))           Theorem 1
+/,(C∘.×D)×X*E∘.+F
```

The final step is based on the fact that `(X*A)×(X*B) ↔ X*A+B`.

A proof in which every step is fully justified is called a **formal** proof; a step which is justified less formally by the observation of some general pattern is called an **informal** proof. We will now illustrate an informal proof by assigning values to the arguments `C` and `D` and displaying the tables `C∘.×D` and `E∘.+F` occurring in the last line of theorem 3:

```
      C←3 1 4                               E←⍳⍴C
      D←2 0 3 1                             F←⍳⍴D
      C∘.×D                                 E∘.+F
6  0  9  3                          0 1 2 3
2  0  3  1                          1 2 3 4
8  0 12  4                          2 3 4 5
```

Since the elements of `E∘.+F` are exponents of `X`, and since the `I`th diagonal of `E∘.+F` (beginning with the zeroth) has the values `I`, each element of the `I`th diagonal of `C∘.×D` is multiplied by `X*I`. We may therefore conclude (informally) that the expression is equivalent to a polynomial whose coefficient vector is formed by summing the diagonals of `C∘.×D`. Using theorem 3 as well, we therefore conclude that this polynomial is equivalent to the product of the polynomials `C P ω` and `D P ω`.

Many useful identities concern what are called (in **APL Language** [7]) **structural** and **selection** functions, such as reshape, transpose, indexing, and compression. For example, a succession of dyadic transpositions can be reduced to a single equivalent transposition by the following identity:

```
I⍉J⍉A ↔ I[J]⍉A
```

The proof is given in Iverson [5]. Further examples of proofs in APL may be found in Orth [8] and in Iverson [1,4].

6. Recursive Definition

A function can sometimes be defined very neatly by using it in its own definition. For example, the factorial function `F:×/1+⍳ω` could be defined alternatively by saying that `F ω ↔ ω×F ω-1` and giving the auxiliary information that in the case `ω=0` the value of the function is `1`. Such a definition which utilizes the function being defined is called a **recursive** definition.

The direct definition form as defined in Iverson [4] permits a "conditional" definition such as:

```
G:ω:ω<0:-ω
```

Such a definition includes three expressions separated by colons and is interpreted by executing the middle one, then executing the first or the last, according to whether the value of the (first element of the) middle one is zero or not. Thus `G ω` is (for scalar arguments) equivalent to `|ω`.

This conditional form is convenient for making recursive definitions. For example, the factorial function discussed above could be defined as `F:ω×Fω-1:ω=0:1`, and a function to generate the binomial coefficients of a given order could be defined recursively as:

```
BC:(Z,0)+0,Z←BCω-1:ω=0:1
```

For example

```
      BC 2              BC 3                       BC 4
1 2 1          1 3 3 1                   1 4 6 4 1
```

Recursive definition can be an extremely useful tool, but one that may require considerable effort to assimilate. The study of existing recursive definitions (as in Chapters 7 and 8 of Orth [8] and Chapter 10 of Iverson [4]) may prove helpful. Perhaps the best way to grasp a particular definition is to execute it in detail for a few simple cases, either manually or on the computer. The details of computer execution can usually be suitably exhibited by inserting ⎕← at one or more points in the definition. We might, for example, modify and execute the binomial coefficient function *BC* as follows:

```
      BC:(Z,0)+0,Z←⎕←BCω-1:ω=0:1

      Q←BC 3
1
1 1
1 2 1
      Q
1 3 3 1
```

We will now give two less trivial recursive definitions for study. The first generates all permutations of a specified order as follows:

```
      PER:(-⌊(⍳!ω)÷!X)⌽X,((!ω),X)⍴PERX←ω-1:ω=1:1 1⍴0

      PER 3                                ⌽'ABCD'[PER 4]
 2 1 0                               DDDDDDABBACCBACCABCCABBA
 2 0 1                               CCABBADDDDDDABBACCBACCAB
 0 2 1                               BACCABCCABBADDDDDDABBACC
 1 2 0                               ABBACCBACCABCCABBADDDDDD
 1 0 2
 0 1 2
```

The second is a solution of the "topological sort" problem discussed on pages 258-268 of Knuth [9]. Briefly stated, an *N* by *N* boolean matrix can specify "precedences" required in the ordering of *N* items (which may represent the steps to be carried out in some production process). If the positions of the 1's in row *I* indicate which items must precede item *I*, then the function:

```
      PR:α[⍋(-⍴α)↑S] PR S⌿S/ω:∧/S←∨/ω:(-1↑⍴ω)↓α
```

provides a solution in the sense that it permutes its vector left argument to satisfy the constraints imposed by the matrix right argument. For example:

```
      C←'ATSFX'
      M                C PR M         PROC            PROC[(⍳5)PR M;]
0 1 0 1 1   TFXAS            ADDRESS     TEXT
0 0 0 0 0                    TEXT        FIGURES
0 1 0 1 1                    STAMP       XEROX
0 0 0 0 0                    FIGURES     ADDRESS
0 1 0 1 0                    XEROX       STAMP
```

If the required orderings among certain items are inconsistent and cannot be satisfied, they are suppressed from the result.

7. Properties of Defined Functions

Defined functions used as building blocks in the development of a complex system play much the same role as primitives, and the comments made on the assimilation of primitives apply equally to such defined functions. Moreover, a clear understanding of the properties of functions under design may contribute to their design.

Many of the general properties of primitives (such as their systematic extension to arrays and the existence of primitive inverse functions) are also useful in defined functions and should be preserved as much as possible. The section on generality addressed certain aspects of this, and we now briefly address some others, including choice of names, application of operators, and the provision of inverse functions.

The names of primitive functions are graphic symbols, and the ease of distinguishing them from the names of arguments contributes to the readability of expressions. It is also possible to adopt naming schemes which distinguish defined functions from arguments, or which even distinguish several sub-classes of defined functions. The choice of mnemonic names for functions can also contribute to clarity; the use of the direct form of definition properly focusses attention on the choice of function names rather than on the choice of argument names.

Present APL implementations limit the application of operators (such as reduction and inner product) to primitive functions, and do not allow the use of defined functions in expressions such as *F/* and *∘.F*. For any defined function *F* it is sometimes useful (although questions of efficiency may limit the usefulness to experimentation rather than general use) to define a corresponding outer product function *OPF*, and a corresponding reduction function *RF*. For example:

```
       F:α+÷ω
     OPF:(α∘.+0×ω) F (α×0)∘.+ω
      RF:(1↑ω) F RF 1↓ω:1=ρω:ω[0]

      A←3 7 11
      B←2 5 10

      A F B              A OPF B               RF B
3.5 7.2 11.1        3.5  3.2  3.1        2.196078431
                    7.5  7.2  7.1
                   11.5 11.2 11.1
```

The importance of inverse functions in mathematics is indicated in part by the number of inverse pairs of functions provided, such as the pair *K*○ω and (-*K*)○ω, the pair *B*⍟ω and *B*⋆ω, and the pair ω⋆*N* and ω⋆÷*N*. Their importance in non-numeric applications is not so commonly recognized, and it is well to keep the matter in mind in designing functions. For example, in designing functions *GET* and *PUT* for accessing files, it is advantageous to design them as inverses in the sense that the expression *K PUT K GET 'FILENAME'* will produce no change in the file.

Other examples of useful inverse pairs include the permutations `ω[P]` and `ω[⍋P]` defined by a given permutation vector `P`, the classification function `C:ω[⍋α]` and its inverse (discussed in Section 4) `CI:ω[⍋⍋α]`, and the "cumulative sum" or "integration" function `CS` and its inverse, the "difference function" `DF` defined as follows:

```
      CS:+\ω
      DF:ω-0,¯1↓ω

      A←3 5 7 11 13 17
      CS A                          DF A
3 8 15 26 39 56               3 2 2 4 2 4
      DF CS A                       CS DF A
3 5 7 11 13 17                3 5 7 11 13 17
```

8. Efficiency

Emphasis on clarity of expression in designing a system may contribute greatly to its efficiency by leading to the choice of a superior overall approach, but it may also lead to solutions which violate the space constraints of a particular implementation or make ineffective use of the facilities which it provides. It is therefore necessary at some point to consider the characteristics of the particular implementation to be used. The speed and space characteristics of the various implementations of APL are too varied to be considered here. There are, however, a number of identities which are of rather general use.

Expressions involving inner and outer products often lead to space requirements which can be alleviated by partitioning the arguments. For example, if `A` and `B` are vectors and `R←A∘.f B`, then the `M` by `N` segment of the result represented by `(M,N)↑R` can be computed as `(M↑A)∘.f (N↑B)`, and `M` and `N` can be chosen to make the best use of available space. The resulting segments may be stored in files or, if the subsequent expressions to be applied to the result permit it, they may be applied to the segments. For example, if the complete expression is `+/A∘.fB`, then each of the segments may be summed as they are produced. Expressions of the form `(M,N)↑R` can also be generalized to apply to higher rank arrays and to select any desired rectangular segment.

If `X` is a vector, the reduction `+/X` can be partitioned by use of the identity:

```
      +/X ↔ (+/K↑X)+(+/K↓X)
```

and this identity applies more generally for reduction by any associative function `F`. Moreover, this identity provides the basis for the partitioning of inner products, a generalization of the partitioning used in matrix algebra which is discussed more fully in Iverson [6].

The direct use of the distribution function `DIS` of Section 3 for summarization (in the form `(DIS A)+.×C`) may lead to excessive use of both time and space. Such problems can often be alleviated in a general way by the use of sorting. For example, the expression `R←A[P←⍋A]` produces an ordered list of the account numbers in which all repetitions of any one account number are adjacent. The points of change in account numbers are therefore given by the boolean vector `B←R≠¯1⌽R` and if the costs `C` are ordered similarly by `S←C[P]`, then the summarization may be performed by summing over the intervals of `S` marked off by `B`.

The sorting process discussed above may itself be partitioned, and the subsequent summarization steps may, for reasons of efficiency, be incorporated directly in the sorting process. Many of the uses of sorting in data processing are in fact obvious or disguised realizations of some classification problem, and a simpler statement of the essential process may lead simply to different efficient realizations appropriate to different implementations of APL.

Like the inner and outer product, recursive definitions often make excessive demands on space. In some cases, as in the function *PER* discussed in Section 6, the size of the arguments to which the function is successively applied decreases so rapidly that the recursive definition does not greatly increase the space requirements. In others, as in the function *PR* of Section 6, the space requirements may be excessive, and the recursive definition can be translated (usually in a straightforward manner) into a more space-efficient iterative program. For example, the following non-recursive definition is such a translation of the function *PR*:

```
X←A PRN W
L1:→(∧/S←∨/W)/L2
A←A[⍋(-ρA)↑S]
W←S⌿S/W
→L1
L2:Z←(-1↑ρW)↓A
```

9. Reading

Perhaps the most important habit in the development of good style in a language remains to be mentioned, the habit of critical reading. Such reading should not be limited to collections of well-turned and useful phrases, such as Bartlett's Quotations or the collections of References 2 and 3, nor should it be limited to topics in a reader's particular speciality.

Manuals and other books **about** a language are, like grammars and dictionaries in natural language, essential, but reading should not be confined to them. Emphasis should be placed rather on the reading of books which use the language in the treatment of other topics, as in the references already cited, in Berry et al [10,11], in Blaauw [12], and in Spence [13].

The APL neophyte should not be dissuaded from reading by the occurrence of long expressions whose meanings are not immediately clear; because the sequence of execution is clear and unambiguous, the reader can always work through sample executions accurately, either with pencil and paper, with a computer, or both. An example of this is discussed at length in Section 1.1 of Iverson [4].

Moreover, the neophyte need not be dissuaded from reading by the occurrence of some unfamiliar primitives, since all primitives can be summarized (together with examples) in two brief tables (pages 32 and 44 of APL Language [7]), and since these tables are usable after the reading of two short sections: Fundamentals (pages 21-28) and Operators (pages 39-43).

Finally, one may benefit from the critical reading of mediocre writing as well as good; good writing may present new turns of phrase, but mediocre writing may spur the reader to improve upon it.

10. Conclusions

This paper has addressed the question of **style**, the **manner** in which something is said as distinct from the substance. The techniques suggested for fostering good style are analogous to techniques appropriate to natural language: intimate knowledge of vocabulary (primitives) and commonly used phrases (certain defined functions), facility in abstract expression (generality), mastery of a variety of equivalent ways of expressing a matter (identities), a knowledge of techniques for examining and establishing such equivalences (proofs), a precise general method for using an expression in its own definition (recursion), and an emphasis on wide critical reading **in** rather than **about** the language.

If one accepts the importance of good style in APL, then one should consider the implications of these techniques for the teaching of APL. Current courses and textbooks typically follow the inappropriate model set by the teaching of earlier programming languages, which are not so simply structured and not so easy to introduce (as one introduces mathematical notation) in the context of some reasonably elaborate **use** of the language. Moreover, they place little or no emphasis on reading in APL and little on the structure of the language, often confusing, for example, the crucial distinction between operators and functions by using the same term for both. **APL Language** [7] does present this structure, but, being designed for reference, is not itself a sufficient basis for a course.

Appendix A

Translation from Direct to Del Form

The problem of translation from the direct to the del form of function definition is fully discussed in Section 10.4 of Iverson [4], the discussion culminating in a set of translation functions usable (or easily adapted for use) on most implementations of APL. Because it is aimed primarily at an exposition of the translation problem, the functions developed in this presentation leave many secondary problems (such as the avoidance of name conflicts) to the user, and the following translation functions and associated variables may be found more convenient for experimentation with the use of direct definition:

```
D←F9 E;F;I;J;K;Q;⎕IO
→((2|+/E='''')∨∧/ 1 3 ≠+/':' I9 E)/ρD←(2ρ⎕IO←0)ρ''
F←'α X9 ' R̲9 'ω Y9 ' R̲9 E←, 1 1 ↓⎕CR ⎕FX 'Q',' ',[¯0.5],E
F←1↓ρD←(0,-6-+/I)↓(-(3×I)++\I←':' I9 F)⌽⍉(7,ρF)ρ(7×ρF)↑F
D←3⌽(C9[((2⌊2⊥∨/'αω' I9 E),1+I),5;]),⍉D[;0,(I←2+ιF-2),1]
J←((¯1⌽I)∧J←>⍀ 0 ¯1 ⌽'←⎕' I9 E)/K←+\I<0,¯1↓I←E∊A9
K←∨/((-K)⌽I∘.>ι1+⌈/K)[;J-1]
D←D,(F,ρE)↑⍉ 0 ¯2 ↓(K+2×K<1⌽K)⌽' ',E,[0.5] ';'

Z←X R̲9 Y;N
Z←(,((1↑X) I9 Y)∘.≠N↑1)/,Y,((ρY),¯1+N←ρX)ρ1↓X

Z←A I9 B                                 C9
Z←(A∘.=B)∧((ρA),ρB)ρ~2|+\B=''''     Z9←
                                       Y9Z9←
Z9←DEF                                 Y9Z9←X9
Z9←⎕FX F9 ⍞                           )/3→(0=1↑,
                                          →0,0ρZ9←
                                          Z9←


      A9
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZA̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲⎕
```

The foregoing functions were designed more for brevity than clarity; nevertheless the reader who wishes to study the translation process in detail may find it useful to compare them with those of Reference 4.

For serious use of direct definition, one should augment the foregoing with functions which record the definitions presented, display them on demand, and provide for convenient editing. For example, execution of:

```
      DEF
DEFR:0⍴⍎'R̲',Y,'←X',0⍴Y←⎕FX F9 X←⍞
DEFR
```

produces a function *DEFR* which, like *DEF*, fixes the definition of any function *F* presented to it in direct form, but which also records the original definition (for later display or editing) in the associated variable *R̲F*. The display of a desired function could then be produced by the following definition:

```
      DEFR
DISPLAY:⍎,(N∧.=(¯1↑⍴N)↑'R̲',⍞)⌿N←⎕NL 2
```

For example:

```
      DEFR
PLUS:α+ω
      DISPLAY
PLUS
PLUS:α+ω
```

References

1. Iverson, K.E., **Algebra, an Algorithmic Treatment**, APL Press, 1976.
2. Perlis, A.J., and S. Rugaber, **The APL Idiom List**, Research Report 87, Computer Sciences Department, Yale University, 1977.
3. Macklin, D., **The APL Handbook of Techniques**, Form Number S320-5996, IBM Corporation, 1977.
4. Iverson, K.E., **Elementary Analysis**, APL Press, 1976.
5. Iverson, K.E., **Operators and Functions**, Research Report 7091, IBM Corporation, 1978.
6. Iverson, K.E., **An Introduction to APL for Scientists and Engineers**, APL Press, 1976.
7. **APL Language**, Form Number GC26-3847, IBM Corporation.
8. Orth, D.L., **Calculus in a New Key**, APL Press, 1976.
9. Knuth, D.E., **The Art of Computer Programming**, Addison Wesley, 1968.
10. Berry, P.C., J. Bartoli, C. Dell'Aquila, V. Spadavecchia, **APL and Insight**, APL Press, 1978.
11. Berry, P.C., and J. Thorstensen, **Starmap**, APL Press, 1978.
12. Blaauw, G.A., **Digital System Implementation**, Prentice-Hall, 1977.
13. Spence, R.L., **Resistive Circuit Theory**, APL Press, 1974.

THE USE OF APL IN TEACHING COMPUTER SCIENCE

Lawrence Snyder
Department of Computer Science
Yale University
New Haven, Connecticut

> "It is the central thesis of this book that the descriptive and analytic power of an adequate programming language amply repays the considerable effort required for its mastery." (Iverson 1962)

In teaching, the objective is to maximize the ideas and concepts transmitted and minimize the extraneous detail. In computer science, the ideas and concepts (problem solving, recursion, logic of computation, etc.), can be especially difficult and abstract. But the vehicle used to carry these concepts to the students, the programming language, can crush them with picayune details. APL is not immune to this criticism - it requires a knowledge of numerous details. But APL has an expressiveness, compactness, breadth and conversational environment not found in other programming languages, and these features make it a pedagogically more productive language. It is remarkable, therefore, that the use of APL in the undergraduate curriculum has met with so much resistance. The purpose of this essay is to soften that resistance by illustrating APL's importance in getting on to the concepts, and suggesting effective methods for getting past the details and addressing some of the criticism of APL's use in the undergraduate curriculum.

A claim that APL is more pedagogically productive than other languages is difficult to substantiate quantitatively in any convincing way. There are too many variables, too many immeasurable parameters involved in this very subjective matter. So I shall use "Introduction to Computer Science for Majors" (CS221) at Yale as a case study to be analyzed for the contributions of APL. The existence of similar courses on many campuses using other programming languages makes it likely that the reader has a basis for comparison. The comparison is supposed to show -- and my limited knowledge about courses elsewhere suggests it will show -- that more material is covered in this class than in similar classes not using APL. Of course, quantity of material transmitted is not as important as the quantity of material received. Since students learn by doing, the homework assignments are probably the best indicators of what the students learned. We address that subject first.

APL Productivity - Interpreters as Homework Problems

Before discussing the way in which APL increases the class' productivity, we must present a sketch of the course's content.

"Introduction to Computer Science for Majors" (hereafter, the Class) is a typical freshman/sophomore level class that I have taught for five years. Some programming experience is required and most students take it as freshman having had BASIC or FORTRAN in high school. The objective of the Class is to **introduce** (as the title indicates) the broadest range of topics possible so that the student can sensibly choose a source of study for the major, and integrate the material as he takes that curriculum. We are able to present a nontrivial introduction to programming in APL and ALGOL, compilers/interpreters, architecture and machine language, heuristic programming, numerical computation and theoretical topics. The syllabus is divided into 9 units that are briefly described in Illustration 1.

Several remarks are in order. The ELI language discussed in units 5 and 6 is reasonably complex language. Though changed yearly, data objects are usually vectors; it has one page BNF syntax (see appendix A), it has FORTRAN-like functions, and it has a few systems operations like `)VARS`. The homework assignment is to implement an interpreter for ELI. (Actually this is a two and a half week assignment with the symbol table and lexical analysis the first week, syntactic analysis pirated from an earlier assignment, and semantics and control the second week. The "execute" operation is not permitted!) No team work is permitted and most students manage to complete at least a "shaky" **implementation** of about 100-200 lines, including comments.

Illustration 1

Syllabus for the Class

Unit 1: Mechanics of using the system (1 week).

Unit 2: The APL language; introduction to arrays and nearly all of the operations, restricted to one-liners (3 weeks). **Homework**: numerous problems, mostly array based and nonnumercial to develop idioms.

Unit 3: Control Structures; **ifs, whiles, cases** and nonrecursive functions (1 week). **Homework**: interactive applications problems such as games.

Unit 4: Recursion; heuristic search; artificial intelligence applications (1 week). **Homework**: recursive problems; backtracking.

Unit 5: Other data structures; stacks and trees; expression trees; postfix BNF (1 week). **Homework**: infix to postfix parser for an expression language (see Unit 6).

Unit 6: Structure of interpreters and compilers; lexical analysis, semantic and control (2 weeks). **Homework**: Write an interpreter for ELI (a "toy" language).

Unit 7: Programming a scalar language; ALGOL-60 or PASCAL (1 week). **Homework**: numerical problems.

Unit 8: Machine architecture; machine language, operating systems (> week). **Homework**: assembly language program for P.D.P 221.

Unit 9: Complexity; theoretical topics, halting problem (< week).

It appears that the major contribution of APL to the implementation project is first and foremost the expressiveness of its operator repertoire. This is testified to by the fact that the students actually complete the problem in two weeks. The majority of the problem is simple table lookup routines, conversions, etc., and the powerful operators simplify these tasks. (See idioms discussion below.) The other powerful feature is the interactive environment which aids immeasurably in debugging.

While we are arguing for the productivity of APL in the context of the interpreter assignment, let us digress briefly to make a point about productivity in scalar languages. We cite a problem used by Gannon to test the characteristics of type conventions in programming languages. Gannon's paper has little relevance to our discussion beyond the fact that his students used two dialects of the scalar language SIMPL and the solved problem [1]:

> "Subjects were asked to read a series of cards of the form ddcc ... c where d is a digit between 0 and 9 and c is a character and to produce dd copies of the reversed character string. Each subject's program was to read the card, place it in an array to be printed, process the card (i.e., produce dd copies of the reversed characters), place the reversed strings in the output array, and print the contents of the array. The program was also to contain a recursive procedure to reverse the character portion of the card."

(See Illustration 2 for an APL solution and sample output.) Gannon then states:

> "While the program is not difficult, it is not trivial either. The solutions ranged from 48 to 216 lines and 27 to 85 statements in ST[Statically-typed dialect of SIMPL] and from 58 to 297 lines and 31 to 105 statements in NT[Non-typed dialect of SIMPL]."

Subjects were allowed two weeks to solve the problem and required an average of 11.61 runs for ST programs, 19.12 runs for NT programs.

Gannon should have said "not trivial in SIMPL"; it is trivial in APL.

The purpose of citing Gannon's experiment is not to criticize it but to illustrate the complexity of problems that students can be expected to solve in two weeks. (Of course, the students may not have required the full two weeks, but the number of runs suggests that two weeks is realistic.) By contrast, students in the Class program most of an interpreter in that time! I suspect that even in APL the interpreter problem is somewhat more difficult than Gannon's problem, but still most students complete the problem. The number of ideas and concepts presented in the interpreter problem certainly repay the Class for their extra efforts.

As a postscript to this digression, it must be mentioned that the recursive reverse routine is sufficiently simple that it has been used as one of a dozen questions on several midterm exams.

Obviously, the student acquires a clear understanding of the internal operation of interpreters and compilers. But there is an additional benefit. Because the ELI language is an approximation of a large subset of APL, the students have no difficulty imagining a full APL interpreter written in APL. This is, of course, a "universal machine" for APL, and it enables one to give a very quick proof (~ half hour) of the undecidability of the halting problem (Unit 9). Although giving the proof is not the same as explaining its fundamental importance, it does spark an interest in theoretical subjects, and tempers the "anything can be programmed" viewpoint.

Another place that the conversational environment contributes is in Unit 8, architecture/machine language. Here, the students are provided with a simulator for a mythical machine, the Pedagogical Data Processor 221. The Class writes an assembly language program for the P.D.P. 221 using an assembler, editor, loader and debugging aids that are also provided. The simulator displays a "front panel" on the CRT screen of the APL terminal and this is sufficiently realistic to prompt the students to "putter" as if it were their own hobby machine. Though the machine is simple and "clean", the students can learn it interactively in a week.

Illustration 2: APL solution (not necessarily minimal) to Gannon problem with sample.

```
      GANNONPROBLEM
03SPOTS       input
03SPOTS     ⎫
STOPS       ⎬ output
STOPS       ⎪
STOPS       ⎭
00NOTHING     input
00NOTHING     output
<CR>

      ∇GANNONPROBLEM[⎕]∇
    ∇ GANNONPROBLEM
[1]   L:→(0=ρX←⍞)/0
[2]    ((1+Y),ρX)ρX,((ρX)×Y←10⊥¯1+'0123456789'⍳2↑X)ρREVERSE '   ',2↓X
[3]    →L
    ∇

      ∇REVERSE[⎕]∇
    ∇  R←REVERSE Z
[1]   →(1≥ρR←Z)/0
[2]  R←(REVERSE 1↓Z),1↑Z
    ∇
```

Teaching the Language - The Role of Idioms

Although it seems to be widely held that APL requires "considerable effort to master" (see Iverson quote above), I do not entirely agree. It is certainly harder to learn than BASIC, but there are two ways to improve acquisition in addition to the generally acknowledged benefits of interaction: the use of large video display and the teaching of **idioms**.

The video display is simply a gadget that projects the terminal output on a (reflected) screen large enough for the whole class to watch the instructor's terminal session. These are coming into wider use and are invaluable for much of a computer science department's curriculum. In addition to the obvious assets, it is useful for the students to watch the instructor make errors! There are true liabilities as well: it is easier to sleep and harder to take notes in the dimly lit room (a requirement of Yale's setup). More seriously, once a lecture has been prepared for the video device, one is at the mercy of the machine being up, and favorable response time - two requirements that are not always fulfilled. Nevertheless, the device makes a material contribution to teaching.

Idiom is Perlis' term for a construction in a programming language for a "logically" primitive operation for which no language primitive exists. For example,

```
(X≠C)/X

((V⍳V)=⍳⍴V)/V
```

are idiomatic for "remove occurrences of *C* in vector *X*" and "remove duplicates", respectively. Perlis and Rugaber [2] have compiled a list of idioms for APL. Like the expressions above, many of the idioms are probably already included in APL courses. But identifying the concept explicitly is useful since it reminds the instructor that acquisition of the "APL style" is primarily a matter of learning idioms for common operations. (T.P. Holls [3] has recently published a different compilation of idioms, etc.)

One idiomatic data structure treated by Perlis and Rugaber deserves special emphasis even though it is mentioned in many text books. This is the tabular structure where an array is used as a vector, the rows being the elements. For example, Gilman and Rose [4] use the following one-liner to convert a sentence (where words are separated by blanks or commas) into the tabular structure:

```
      T←'THIS STATEMENT IS, I BELIEVE, FALSE'
      (⍴A)⍴(,A←A∘.≥⍳⌈/0,A←(A≠0)/A←A-1+0,¯1↓A←A/⍳⍴A)\(~A←T∊',')/T←T,' '


THIS
STATEMENT
IS
I
BELIEVE
FALSE
```

The tabular structure provides a mechanism for forming vectors whose elements are composites. One can **think** of the structure as a homogeneous vector and hence apply the same program formation techniques used with primitive data objects. The structure requires specialized operators, of course, and development of these as homework assignments is a good way to promote the use of "generalized vectors".

Another mechanism for promoting "APL style" that I believe is very effective is **to force** the use of one-liners. ("One-liners" means non-recursive functions with no occurrences of →.) This is especially true since most of the students in the Class have programmed in a scalar language and they have a great temptation to "write BASIC" in APL. Since all of the APL operators cannot be introduced at once, the students must struggle to create one-liners with a subset of the operators. Thus, by the fifth week (Unit 3) when explicit control is introduced, the "APL style" is sufficiently instilled that it tends to persist. Lectures about "structured programming", though given, are largely unnecessary, since control structure is not generally abused.

A Remark to the Critics

There is little point in attempting a detailed refutation of APL critics, but the Class does illuminate one point.

Generally, the computer science community is still caught up with what Backus [5] calls the "von Neumann" style of programming and there is little doubt that most of the programs executed today are written in scalar languages. This apparently convinces many academicians of the appropriateness of teaching such languages in introductory courses, since the students are thusly "prepared" for the "real world". To me, this is wrong-headed. We are to lead, not follow!

Our experience in the Class has been that competent APL programmers can learn ALGOL-60 in three lectures (including the mechanics of using a different computer). This is probably aided by the fact that the students have usually programmed in some scalar language already, and by the fact that BNF and the structure of languages has been discussed earlier. But the conclusion must be that learning "easier" languages is easier after APL and having taught it first is no disservice. It would be very interesting in this regard to see if a population of APL programmers would do significantly better in Gannon's experiment.

Summary

Experiences in "Introduction to Computer Science for Majors" have been cited to support the claims that (1) APL is sufficiently powerful to permit the treatment of a large number of ideas and concepts, (2) the difficulty of mastering APL can be reduced by several mechanisms, namely, use of idioms, and (3) that the "everyone uses scalar languages, so that's what we should teach" criticism is misguided. Unlike most discussions of faith where believers continue to believe and skeptics remain skeptical, I hope at least a few will have been converted.

Acknowledgements

I owe a great debt to Alan Perlis and Stanley Eisenstat for many useful conversations that have led to many of these ideas.

Appendix A

ELI SYNTAX

1. <dig> ::= 0|1|2|3|4|5|6|7|8|9

2. <integer> ::= <digit> | <integer> <digit>

3. <number> ::= <integer> | ¯<integer>

4. <vector> ::= <number> | <vector> b <number>

5. <letter> ::= *A*|*B*|*C*|*D*|*E*| .... |*Z*

6. <var> ::= <letter> | <var> <letter>

7. <operand> ::= <var> | <vector> | (<expr>)

8a <expr> ::= <var> | <vector>

8b | <expr> + <operand>

8c | <expr> - <operand>

8d | <expr> ⌊ <operand>

8e | <expr> ⌈ <operand>

8f | <expr> / <operand>

8g | <expr> \ <operand>

8h | <expr> ρ <operand>

8i | <expr> ↑ <operand>

8j | <expr> ↓ <operand>

8k | <expr> = <operand>

9. <fdef> ::= ∇ <var> ← <var> b <var> : <expr> ∇

```
10.   <statement> ::= <expr>
                    | <var> ← <expr>
                    | <fdef>
                    | <var> b <var> b <var> → <var>
                    | <var> ←
                    | )ERASE <var>
                    | )VARS
                    | )CLEAR
```

Note: The associated semantics restrict the data objects to vectors.

References

1. Gannon, J.D., "An Experimental Evaluation of Data Type Conventions", CACM (20) 8, pp. 584-595, August 1977.

2. Perlis, A.J. and J.S. Rugaber, "The APL Idiom List", Research Report #87, Yale Department of Computer Science, 1977.

3. Holls, T.P., "APL Programming Guide: Vector Operations", IBM PDP Scientific Marketing, White Plains, New York, 1978.

4. Gilman, L. and A. Rose, **APL: An Interactive Approach**, 2nd Edition, Revised, Wiley, 1976.

5. Backus, J.W., "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs", IBM Research Report RJ 2234, 1978.

THE ARCHITECTURAL ELEGANCE OF CRYSTALS MADE CLEAR BY APL

Donald B. McIntyre
Department of Geology
Pomona College
Claremont, California

Abstract

One reason for the fascination of crystals is their external symmetry, which results from the regularity with which their constituent atoms are arranged. Efficient packing of the atoms is achieved by periodic three-dimensional repetition of a unit cell. Because the faces and edges are related in a simple way to the cell's shape and size, their geometric relations are easily expressed as APL functions. Where the traditional method of computation by spherical trigonometry treated the morphology of each crystal as a special case, APL encourages unification, and brings out the beauty of the crystal structure. Students now start with the most general case and learn to see higher symmetries as successive simplifications of it. Because the Earth's crust is composed of crystals, geology students -- who often have little taste for mathematics -- must study crystallography early in their training. APL gives them mastery of topics that would be otherwise beyond their reach, and they acquire mental and computational skills of wide application.

Introduction

Contrary to common belief, crystals are not exotic rarities, but the very substance of the Earth itself. Apart from natural glasses and organically derived material like coal, all rocks are made up of crystals. The word "crystal", derived from the Greek word for "ice", refers to the long-held belief that the clear crystals of quartz ("rock crystal") discovered in high mountains had formed when water was frozen so hard it would never melt. Until our own time, chemists, recognizing the principle that bodies do not react unless in solution, began every study by dissolving the material to be analyzed; thus they destroyed the crystalline structure at the very start of their investigation.

We now know that, except for beaches of broken sea-shells, every sand grain is a crystal, and that most of them are quartz, with the composition $Si0_2$. What characterizes a crystal is the high degree of order in its internal structure, which is expressed in a symmetrical arrangement of faces if the growing crystal is free from external restraint. A grain of sand is a battered crystal, which in the right chemical surroundings and at the right temperature and pressure, will once again grow the faces it may have formerly possessed. Like an animal, it depends upon its environment, and responds when the environment changes; but, unlike an animal, the crystal is timeless, for it can wait a billion years to get the nourishment it needs in order to grow.

Crystallography is the basis for the study of the solid state, and it is natural that geology students should begin their systematic work with crystallography, because rocks are mixtures of minerals, and minerals are crystals. But to geology students with little love of mathematics, crystallography often appears formal and forbidding. For many years I have used APL as a convenient and appropriate notation in teaching crystallography; originally with no implementation available, then with more or less inconvenient access by terminal, and from 1975 with an IBM 5100 and video monitors in the classroom. The class is at the sophomore level, and there are normally about 12 students. There are no prerequisites; the necessary mathematics, including spherical trigonometry and matrix methods, being developed as needed. Outside the class, students have ready access to the 5100, which has a Tektronix 4662 plotter attached. I use zero index origin throughout the course and in the examples in this paper.

There are many ways to organize the material, and I have probably used a different one each time I have taught the course. It is important always to remember that the subject is crystallography and mineralogy, not mathematics or computer programming. For this reason, I introduce notation only as needed for the work in hand, minimizing the computer and machine characteristics. Indeed, I do not mention APL to start with, treating the primary functions as natural extensions and revisions of algebra, and I use no APL text. The system introduced by Iverson (Iverson, 1976) whereby functions are directly defined (which is the subject of my second paper) greatly assists because it keeps functions to a single line and eliminates the need for a header. The 5100 has the advantage that APL is available as soon as it is switched on, since no sign-on procedures and no billing are involved. The editing and scrolling features get rid of the frustration associated with use of other systems, especially when the user is not a good typist or when he is composing a function.

Close-Packed Layers

A simple way to introduce the principles of crystal structure and to illustrate the relationship between architecture and physical properties is to consider the structure of gold, silver, and copper, in which the atoms are all of one kind and can be considered as spheres of fixed radius. In such minerals the atoms can be efficiently packed as a sheet within which each sphere touches 6 neighbors; this is called a close-packed layer. Another close-packed layer can be placed on top of the first one, but if the packing is to remain efficient, the new layer must be offset, so that its spheres fit into the holes of the layer below. This is the arrangement of atoms in gold, silver, and copper. It also approximates the arrangement of oxygen ions in many minerals.

Because the oxygen ion is larger and more abundant than other common ions, it tends to dominate the structure of those minerals (notably silicates) that make up the Earth's crust. Other ions, being smaller, often fit into the interstices (there are two kinds, of different size) between the close-packed layers of oxygen. Those that are too big to do so destroy the close-packing, and the resulting minerals have lower density; this, in fact, is one reason for the existence of dry land. It is therefore a useful exercise to compute the sizes of the holes between two close-packed layers, and the radius ratio, so determining into which of the two kinds of hole another ion can fit.

Suppose then that we glue ping-pong balls to a board, to make them form a close-packed layer with the outline of an equilateral triangle. If *N* is the number of balls along a side, then enumeration shows that `+/1+⍳N` is the total number of balls in the layer. The expression `+/1+⍳N` suggests `×/1+⍳N`, which, of course, is `!N`. Moreover, using Iverson's notation for equivalent expressions:

```
+/1+⍳N
0.5×+/(⌽1+⍳N)+(1+⍳N)
0.5×N×1+N
0.5×N+N*2
```

For example, with 8 balls on a side, we need 36 balls for the triangular sheet.

Let a third layer be placed on top of the first two, so that the new balls are directly above those on the lowest layer. This 1-2-1 pattern is hexagonal close-packing (hcp), an important one in nature; for example, the arrangement of oxygen in ruby (Al_2O_3). But there is an alternative and more interesting packing. Move the balls of the third layer, which were in one set of holes on top of the second layer, so that they occupy the other set of holes. They are then no longer vertically above the balls of the lowest layer. This 1-2-3 pattern is cubic close-packing (ccp). The unit of pattern is a face-centered cube, oriented in such a way that one of the body diagonals is perpendicular to the layers. This is the arrangement of atoms in copper, and it is the arrangement of chlorine ions in common salt (NaCl). Whereas hcp has only 1 set of close-packed layers, ccp has 4 such sets; hence the malleability and ductility of metals like gold, silver and copper.

Continue to build a ccp arrangement by adding successive layers, each one smaller than its predecessor, until a single ball at the apex completes a tetrahedral (4-faced) structure. Inspection of the model will show that the three sloping faces that have been constructed are close packed, like planes parallel to the base. If we now measure the slope of these faces and the angle between their normals, we should find that within experimental error they are 70.53 degrees and 109.47 degrees respectively, the same as the angles between octahedral faces measured across an edge and across a corner. 109.47 degrees is also the bond-angle for silicon in silicates and for carbon in aliphatic compounds, so it is of fundamental importance for both the organic and inorganic worlds. How can these angles be computed? We pose the problem, but wait until later to give an answer.

How many ping-pong balls do we need in order to make this ccp model? The answer is a function of *N*, the number of balls on one edge of the base, but what is the function? We can use the expression `+/1+⍳N` successively to compute the number of balls in each layer from `N←0` to `N←8` and build the vector *A*:

```
0  1  3  6  10  15  21  28  36
```

Then the total number of balls needed for successively larger values of *N* is `B←+\A`

```
      B
0  1  4  10  20  35  56  84  120
```

Let *PW* be defined as the pairwise function

```
      PW:⍎2⌽' YX ',α,0ρX←1↓ω,0ρY←¯1↓ω
```

Then `D←'-' PW B` gives the successive differences of *B*

```
      D
1  3  6  10  15  21  28  36
      '-' PW '-' PW B
2  3  4  5  6  7  8  9
      '-' PW '-' PW '-' PW B
1  1  1  1  1  1  1
```

It should be noted that the inverse of `'-' PW B` is

`+\B[0],D` (where `B[0]` is like a constant of integration) and not

```
      '+' PW D
```

Consequently the total number of balls needed for successively larger models is

```
      +\+\+\Nρ1
      +\+\1+ιN
```

Thus:

```
      CCP:+\+\1+ιω

      CCP 12
1  4  10  20  35  56  84  120  165  220  286  364
```

Although the problem and its solution may seem trivial, experience shows that they are by no means so to beginners, and much is learned by a student who struggles with them. Another solution offered was

```
      CPACK:+/+\(I∘.≤I)×(-ιω)⌽(ω,ω)ρI←1+ιω

      CPACK 12
364  286  220  165  120  84  56  35  20  10  4  1
```

This is interesting because it involves essentially building a computer model of the successive layers, but it fails to recognize and take advantage of the symmetry of the successive differences.

As *CCP* makes clear, the second differences are an arithmetic progression, and the third differences are therefore constant. *CCP*, in fact, directs our attention to the existence of a third degree polynomial whose solution would give the number of balls needed for a model corresponding to a given value of *N*. Taking the values for `N←0 1 2 3`, we determine the coefficients, expressing the result as

```
      CCP2:(ω∘.*1 2 3)+.÷3 2 6

      (CCP2 1+ι12)∧.=CCP 12
1
```

We started the discussion of close-packed layers by noting the analogy between the expression for the number of balls in a layer (`+/1+ιN`) and the factorial (`×/1+ιN`). The resemblance is also seen when the functions are defined recursively using Iverson's notation for direct definition:

```
      FAC:ω×FACω-1:ω=1: 1

      NBL:ω+NBLω-1:ω=1: 1
```

Thus:

```
      (FAC 8)=!8
1
```

and:

```
      (FAC 8)=×/1+⍳8
1
      (NBL 8)=+/1+⍳8
1
```

The advantage gained by a student who pursues the example thus far is that he discovers how the notation can be used as an aid for thinking about his problem, which he learns to see in new and different ways. While tackling a problem meaningful to him, he has had an introduction to APL that he can build upon and use in other contexts.

The Crystal Lattice

The periodic repetition characteristic of all crystals is illustrated in the model of the structure of copper described in the previous section. In order to understand the geometry involved and develop the notation used by crystallographers, we take first a 2-dimensional example. Our framework of reference is a set of Cartesian axes, mutually orthogonal and with equal metrics. In 3-dimensions we use a right-handed set of axes with +X towards us, +Y to the right, and +Z pointing upwards. For a 2-dimensional example we suppose that we are looking down on the XY plane. The following text supposes that the reader sketches appropriate illustrations.

Let *D* be a matrix whose rows define vectors *A* and *B* with respect to axes X and Y.

```
      D
2       1.25
0.5     1.75

      A←D[0;]
      B←D[1;]
```

The student is asked to verify **both geometrically and algebraically** that

A+A
2×A

and that

A+A+A+A
4×A

Having established the meaning of multiplication by a scalar, the student can then define a row of lattice points:

```
      A∘.×⍳6
   0         2         4         6         8         10
   0         1.25      2.5       3.75      5         6.25
```

Crystallographers use the notation [u v w] to denote the vector (u×A) + (v×B) + (w×C); that is, u units along the A-axis, v units along the B-axis, and w units along the C-axis. When A, B, C are properly chosen (they are referred to as the crystallographic axes **a**, **b**, **c**, as opposed to the Cartesian axes X, Y, Z), u v w are found to be simple integers for all edges defined by the intersection of naturally occurring crystal faces. Thus, in the 2-dimensional example the vector `[2 3]` is

```
      (2×A) + (3×B)
      2  3 +.× D
      5.5  7.75
```

Thus *D* serves to convert crystallographic vectors [u v w], which are simple integers, into Cartesian vectors, which are more convenient for calculation of lengths and angles. The inverse of *D* can, of course, be used to convert Cartesian vectors back to crystallographic; for example:

```
      C←1.7  3.2+.×D
      C
5  7.725

      C+.×⌹D
1.7  3.2

      C⌹⍉D
1.7  3.2
```

Having understood the geometrical significance of addition and scalar multiplication of vectors, the student has grasped what is meant by linear combinations, and he realizes that in 3-dimensions the linear combination of two vectors will give a third vector that lies in the plane of the other two. In crystallography when 3 vectors are co-planar, they are said to define a **zone** and be co-zonal. This is a common occurrence for which tests must often be made. An obvious method is to reshape the 3 vectors into the rows of a matrix, *M*; then `⌹M` will give a domain error if the vectors are co-zonal.

A 2-dimensional lattice net is generated by adding points from one lattice row to points in another. Thus:

```
      M←(A∘.×⍳6)∘.+(B∘.×⍳4)
```

Looking at the shape of *M*,

```
      ⍴M
2  6  2  4
```

the student is expected to understand where these numbers come from, and particularly to see the significance of the two 2's. The outer "product" `∘.+` has combined **every** element of its first argument with **every** element of its second; which means that not only have X's been added to X's, and Y's to Y's, but also X's have been added to Y's and Y's to X's, which has no meaning in crystallography. To eliminate the unwanted values we need a 3-dimensional selection from this 4-dimensional array:

```
      ⍴0  1  0  2⍉M
2  6  4
```

It would, however, be convenient to change the order of the axes, so that the axis of length 2 (corresponding to X and Y) will be last rather than first:

```
      ⍴2  0  1⍉0  1  0  2⍉M
6  4  2
```

This could, of course, have been achieved in a single step by an appropriate left argument for a single dyadic transposition:

```
      ⍴2 0 2 1⍉M
6 4 2
```

Observing that

```
      2 0 1[0 1 0 2]
2 0 2 1
```

we can define

```
      I←2 0 1
```

and

```
      J←0 1 0 2
```

and generalize as follows:

```
      ⍴I⍉J⍉M
6 4 2

      ⍴I[J]⍉M
6 4 2
```

Within the first week of the class, the students are able to extend this procedure to a 3-dimensional crystal lattice, and to write down the appropriate left argument for the dyadic transpose of the resulting 6-dimensional array. The appropriateness of the notation to a task in crystallography that is already thoroughly understood makes a trifle of what would appear to be rather frightening if presented as an abstract problem without motivation or context.

Because the encode function ⊤ is useful in many unexpected situations, it is appropriate to show how it provides an alternative way of generating the points of a 2-dimensional net:

```
      ⍴(⍉D)+.×0 10⊤⍳100
2 100
```

Finally, once a student has understood both matrix multiplication and dyadic transposition, it is hardly a digression to show him that if

```
      M←?4 3⍴20
```

and

```
      N←?3 5⍴20
```

then

```
      ∧/,(M+.×N)=+/0 2 2 1⍉M∘.×N
1
```

Angles of the Octahedron

Because crystallographic axes are edges of the unit cell in a crystal lattice, they may be oblique to one another and have different lengths. Consequently, in order to compute vector lengths and angles, it is in general necessary to transform the reference system from crystallographic to Cartesian. But the octahedron is a special case; it has cubic symmetry and therefore its crystallographic axes are already Cartesian. With respect to these axes, the normals to the octahedral faces are:

$1\ 1\ 1$ $\qquad$ $1\ 1\ \bar{1}$

$1\ \bar{1}\ 1$ $\qquad$ $1\ \bar{1}\ \bar{1}$

$\bar{1}\ 1\ 1$ $\qquad$ $\bar{1}\ 1\ \bar{1}$

$\bar{1}\ \bar{1}\ 1$ $\qquad$ $\bar{1}\ \bar{1}\ \bar{1}$

Crystallographers denote a negative component in a vector by writing a bar over the corresponding digit, and because only single digit indices occur, no ambiguity results if the indices are written without intervening spaces or punctuation. APL is the only computing language that meets this need for notation; its raised negative sign, and the intervening spaces, allow generalization to components of more than one digit.

If the rows of *P* represent the poles (normals) to the front face of the cube and 3 of the octahedral faces:

```
      P
 1  0  0
 1  1  1
¯1 ¯1  1
 1 ¯1  1
```

these vectors are normalized to unit length by the expression:

```
      P←P÷((+/P*2)*.5)∘.+3⍴0
```

Then the matrix of angles between these vectors is:

```
      X←(¯2○P+.×⍉P)×180÷○1
      X
      0            54.7356      125.264     54.7356
     54.7356        0           109.471     70.5288
    125.264       109.471        0          70.5288
     54.7356       70.5288       70.5288     0
```

Consequently the angle from the cube to the octahedron in degrees, minutes, and seconds is:

```
      0 60 60⊤60×60×X[0;1]
54  44  8.19714
```

To convert all the angles from degrees and decimals to degrees, minutes, and seconds:

```
      Y←0 60 60⊤60×60×X
      ⍴Y
3  4  4
```

Or rearranging into a more convenient form:

```
      2 0 1⍉0 60 60⊤60×60×X
    0          0          0
   54         44          8.19714
  125         15         51.8029
   54         44          8.19714

   54         44          8.19714
    0          0          0
  109         28         16.3943
   70         31         43.6057

  125         15         51.8029
  109         28         16.3943
    0          0          0
   70         31         43.6057

   54         44          8.19714
   70         31         43.6057
   70         31         43.6057
    0          0          0
```

Bond Lengths and Angles

The minerals cassiterite (SnO_2), rutile (TiO_2), and pyrolusite (MnO_2) are of economic and strategic importance as primary sources of tin, titanium, and manganese. They also have the same crystal structure. In each case the metal atoms are at the corners and at the center of a tetragonal cell, which for cassiterite has edges of 4.74 4.74 3.19 Angstroms. The matrix defining the direct lattice, whose rows are the crystallographic axes with respect to the Cartesian, is a diagonal matrix, because tetragonal crystals have orthogonal axes.

```
      D
 4.74      0         0
 0         4.74      0
 0         0         3.19
```

With respect to this cell the tin atoms are at 0 0 0 and 0.5 0.5 0.5, and the oxygen atoms are at:

```
      A
 0.2         0.2         0.5
 0.8         0.8         0.5
 0.7         0.3         0
 0.3         0.7         0
 0.7         0.3         1
 0.3         0.7         1
```

Moving the origin to the central tin atom, and converting to Cartesian:

```
      A←A-.5
      A←A+.×D
```

Then the bonds radiating from the tin atom have the lengths:

```
      L←(+/A*2)*.5

      L
2.011  2.011  2.0836  2.0836  2.0836  2.0836
```

After normalizing the bonds to unit length, we can print a table of bond angles as seen from the central tin atom:

```
      A←A÷L∘.+3⍴0

      9 2⍕(¯2○A+.×⍉A)×180÷○1
      .00   180.00    90.00    90.00    90.00    90.00
   180.00      .00    90.00    90.00    90.00    90.00
    90.00    90.00      .00    80.10    99.90   180.00
    90.00    90.00    80.10      .00   180.00    99.90
    90.00    90.00    99.90   180.00      .00    80.10
    90.00    90.00   180.00    99.90    80.10      .00
```

Triclinic Crystals

The Triclinic System is the general case in crystallography. The axes (cell edges) are all unequal, and no two are mutually perpendicular. An example is chalcanthite, $CuSO_4.5H_2O$, which is easily grown into beautiful blue crystals.

The matrix whose rows are the Cartesian coordinates of the crystallographic axes I designate *D*, for "direct" lattice. For chalcanthite this matrix is:

```
      D
     5.82934     0           ¯1.83053
     2.00205     10.3847     ¯1.43619
     0           0            5.95
```

The crystallographic axes are then:

```
      NORM D
6.11  10.673  5.95
```

Where *NORM* gives the lengths of Cartesian vectors

```
      NORM:(+/⍵*2)*.5
```

The interaxial angles, designated in crystallography as α, β, and γ, are given in the matrix

```
      (.001<|A)×A←D ANGLE D
      0          77.333      107.43
     77.333       0           97.733
    107.43       97.733        0
```

Where *ANGLE* gives the angle in degrees between Cartesian vectors:

```
      ANGLE:DFR ¯2○1⌊¯1⌈(DIR⍺)+.×⍉DIR⍵
```

DIR (for **Dir**ection vector) normalizes a vector; that is, cuts off a segment with unit length:

```
      DIR:⍵÷(NORM⍵)∘.+(0⊥⍴⍵)⍴0
```

DFR computes **d**egrees **f**rom **r**adians:

```
      DFR:⍵×180÷○1
```

The volume of the cell is:

```
      VOL D
360.189
```

Where *VOL* is the vector triple product:

```
      VOL:⍵[0;]+.×⍵[1;]CP⍵[2;]
```

and *CP* is the vector cross product:

```
      CP:((1⌽⍺)×¯1⌽⍵)-(¯1⌽⍺)×1⌽⍵
```

However, the volume of the cell is also, and more neatly, given by the determinant

```
      DET D
360.189
```

where *DET* is defined as:

```
      DET:-/+/×/[1](1 ¯1∘.×⍳3)⌽0 0∘.+⍵
```

Although the shape, size, and volume of the unit cell are easily computed from the *D*-matrix, crystallographic reference books give instead the axial lengths (**a b c**) and angles (α β γ). This is unfortunate, because the *D*-matrix is the key that transforms vectors in the direct lattice to Cartesian representation. Moreover, although axial lengths and angles are found very simply from *D*, the inverse problem has an awkward solution.

If the axial lengths and angles are the rows of a matrix:

```
      CHALCANTHITE
     6.11        10.673        5.95
    97.7333     107.433       77.3333

      D←DMAT CHALCANTHITE
```

Where:

```
DMAT:(⍉3 3⍴⍵[0;])×3 3⍴(DA RFD⍵[1;]),(DB RFD⍵[1;]),0 0 1
```

```
DA:1 0 1\1 2○⍵[1]
```

```
DB:((1○⍵[0]×C,0○C←COSC⍵),2○⍵[0]
```

for the **a** and **b** axes.

RFD for **r**adians **f**rom **d**egrees:

```
RFD:○⍵÷180
```

P
COSC gives the **cos**ine of angle C in a spherical triangle whose sides are known:

```
COSC:¯1↑CINC 1 3⍴⍵
```

CINC gives the **c**osines of the three **inc**luded angles, A B C, of a spherical triangle whose sides are known:

```
CINC:(-/(2○X)×.*L)÷×/0 1↓(1○X←1 2 3⌽3 3⍴⍵)×.*L←3 PR 2
```

PR is a logical matrix used to **par**tition another matrix into subsets:

```
PR:X,[.5]~X←(⍳α)∊⍵
```

Miller Indices and the Reciprocal Lattice

Miller's rule for assigning the indices (**h k l**) to a plane parallel to an actual or possible crystal face is that the intercepts it makes on the crystallographic axes are (**a**,**b**,**c**)÷(**h**,**k**,**l**). The fact that **h k** and **l** are always found to be simple rational numbers is called the Law of Rational Indices, and is one of the fundamental observations of crystallography.

This "reciprocal of intercepts" method always appears confusing to a student, because, as defined, the Miller indices (**h k l**) do not represent a vector and lack the useful properties possessed by vectors. But if we make a new lattice, the **reciprocal lattice**, by a linear transformation of the direct lattice, then (**h k l**) becomes a true vector. Its direction is that of the face normal, and its length is the reciprocal of the **d**-spacing; that is, of the spacing between planes in the set.

Although the idea originated with Bravais in 1850, its power was not generally recognized until the development of X-ray diffraction, which makes great use of **d**-spacings of the lattice planes. Having no way to determine the dimensions of the lattice, Bravais used a **polar** lattice, in which the axes were the normals (poles) of the faces.

If

```
      CHALCANTHITE
 6.11          10.673          5.95
97.7333       107.433         77.3333
```

then

```
      MFD POLAR CHALCANTHITE[1;]
      85             73            100
      45.429         46.653         54.724
```

where

```
        POLAR:180-DFR ¯2○CINC RFDω
```

MFD **m**inutes **f**rom **d**ecimals of degrees

```
        MFD:(0 60⊤|ω×60)×1 1∘.××ω
```

and its inverse, *DFM* **d**ecimals **f**rom **m**inutes

```
        DFM:(60⊥ω)÷60
```

If *D* is the matrix that defines the direct lattice, then `R←⍉⌹D` defines the reciprocal lattice. That this is so can be seen as follows:

If

```
      R
      ⍉⌹D
```

then

```
      D+.×⍉R
      (⍳3)∘.=⍳3
```

The off-diagonal zeros result because the axes of the reciprocal lattice, **a* b* c***, being face normals, are by definition perpendicular to axes of the direct lattice taken in pairs; thus, **a*** is perpendicular to the plane of axes **b** and **c**, and so on.

Because the lengths of the axes of the reciprocal lattice are defined as the reciprocals of the **d**-spacings of corresponding planes, and as the **d**-spacing of (1 0 0) is the projection of **a** on to the normal to (1 0 0), it follows that the diagonals of the matrix must be 1.

Consequently, if

```
        R←⍉⌹DMAT CHALCANTHITE
```

Then `NORM R` gives the lengths of **a* b* c***; that is, the **d**-spacings of (1 0 0), (0 1 0), and (0 0 1).

If the Miller indices of faces on a chalcanthite crystal are given by *F*, where:

```
      F
1  0  0
0  1  0
0  0  1
1  3  0
1  2  0
1  1  1
0 ¯2  1
```

the angles between them are

```
      9 2⍕(F+.×R) ANGLE F+.×R

        .00    100.91     73.78     67.07     53.83     44.88     70.05
     100.91       .00     85.76     33.85     47.08     74.77    135.54
      73.78     85.76       .00     76.83     74.38     36.81     49.78
      67.07     33.85     76.83       .00     13.24     49.59    118.42
      53.83     47.08     74.38     13.24       .00     41.90    109.41
      44.88     74.77     36.81     49.59     41.90       .00     68.83
      70.05    135.54     49.78    118.42    109.41     68.83       .00
```

Computations of **d**-spacings for a triclinic crystal is shown for the example of the rare mineral betaroselite (calcium cobalt arsenate hydrate), which is numbered 17-166 in the file published by the Joint Committee on X-ray Powder Diffraction Standards.

The axial lengths are 5.88 7.67 and 5.58 Angstroms, and the axial angles are 112° 19′, 17° 21′, and 119° 41′. What are the **d**-spacings for the following planes, identified by their Miller indices?

```
      F
 0  1  0
 0  0  1
 1 ¯1  1
¯2  1  1
 1  1 ¯2

      BETAROSELITE←DFM ⍉3 2⍴112 19 71 21 119 41
      BETAROSELITE←2 3⍴5.88 7.67 5.58,BETAROSELITE

      BETAROSELITE
    5.88           7.67           5.58
  112.317         71.35         119.683

      R←⍉⌹DMAT BETAROSELITE
```

Thus the table of **d**-spacings is given by

```
      F,÷NORM F+.×R
 0     1     0      6.41804
 0     0     1      5.09223
 1    ¯1     1      4.63154
¯2     1     1      2.18413
 1     1    ¯2      2.05447
```

Even advanced texts recommend graphical methods to determine the d-spacings of triclinic crystals. But APL makes the solutions so simple that we can begin with the triclinic as the general case and consider higher symmetries as progressively more specialized. Students also feel a sense of accomplishment when they discover they are able to find mistakes in standard reference works.

Transformations

The unit cell is always chosen with regard to the symmetry elements present, but when there is insuffcent symmetry the axes selected are usually the 3 shortest vectors from any lattice point to its neighbours. The problem may seem trivial, but the literature is full of examples of monoclinic and triclinic crystals whose cells have not been chosen consistently. Chalcanthite, which is triclinic, is a good example, apparently because attempts were made to select a cell that would give simple indices to conspicuous faces.

Gypsum, $CaSO_4.2H_2O$, makes an excellent illustration, because although its monoclinic symmetry fixes the **b**-axis normal to the mirror plane, no symmetry dictates the orientation of the other axes within that plane, and as many as 5 different choices have been published. In the **a-c** plane, each lattice point has 6 almost equidistant neighbors approximating a hexagon. Because the problem is in 2-dimensions it is well suited for detailed study.

Consider two of these proposed cells in the **a-c** plane: the cell chosen by Dana, on the basis of external morphology, has one lattice point within it, whereas De Jong's cell is primitive, with the shortest possible vectors as edges. They have the same **c**-axis, but the **b**-axis goes in opposite directions in the two cells.

Let *T* be the matrix that transforms vectors [u v w] in the direct lattice from Dana's axes to De Jong's, then

```
      J←D+.×T
```

where *D* is a set of non-coplanar vectors according to Dana, and *J* is the same set according to De Jong. If we can write down De Jong's coordinates [u v w] for what in Dana's system would be [1 0 0] [0 1 0] and [0 0 1], then the matrix *D* in this expression is the identity matrix and

```
      T←J
```

Inspection of the **a-c** plane of the lattice leads to the result

```
      T
¯2  0 ¯1
 0 ¯1  0
 0  0  1
```

T can now be used to convert **any** vector from Dana's system of coordinates to De Jong's. The inverse of *T* will, of course, make the conversion back again:

```
      ⌹T
¯0.5    0    ¯0.5
 0     ¯1     0
 0      0     1
```

The determinant of *T* gives the relative sizes of the two cells. This is because the determinant of a product is the product of the determinants, and the determinant of a cell matrix gives, as we have seen, the volume of the cell. Cells are always outlined by lattice points at their corners, consequently the determinant of *T* must be an integer, or the reciprocal of an integer.

```
      DET T
2
```

Dana's cell is twice as big as De Jong's.

If *T* is the transformation matrix for the direct lattice, then ⍉⌹*T* is the transformation matrix for the reciprocal lattice, which must be used to transform Miller indices (h k l).

```
      ⍉⌹T
¯0.5    0     0
 0     ¯1     0
¯0.5    0     1
```

This matrix has an eigenvector 1 0 ¯3 corresponding to the eigenvalue of 1. This means that the face whose Miller indices are (1 0 ¯3) according to Dana has the same indices according to De Jong.

Even when it is not possible to write down *T* by inspection, it can be computed if any 3 vectors that are not in one plane can be given indices in two coordinate systems. For example, the file published by the Joint Committee on X-ray Powder Diffraction Standards has two different discriptions of gypsum (J6-0046 and J21-816). Because **d**-spacings and intensities are given for these patterns it is possible to equate the following planes:

J6	J21
0 2 0	0 2 0
0 3 1	1 3 0
2 0 0	2 0 ¯2

Thus if

```
      J21←J6+.×T
      T←(⌹J6)+.×J21

      0⍕T
 1  0  ¯1
 0  1   0
 1  0   0

      0⍕J6A,J6A+.×T
   J6A      J21A

 0  2  0   0  2  0
 1  2 ¯1   0  2 ¯1
 0  3  1   1  3  0
 1  1 ¯2  ¯1  1 ¯1
 1  4 ¯1   0  4 ¯1
 0  0  2   2  0  0
 2  1 ¯1   1  1 ¯2
 0  5  1   1  5  0
 1  5  0   1  5 ¯1
 2  0 ¯2   0  0 ¯2
 2  0  0   2  0 ¯2
 2  2 ¯2   0  2 ¯2
 1  4  1   2  4 ¯1
 1  5 ¯2  ¯1  5 ¯1
 2  4 ¯2   0  4 ¯2
 1  2 ¯3  ¯2  2 ¯1
 1  0  0   1  0 ¯1
 0  0  1   1  0  0
 1  0  1   2  0 ¯1
 1  0 ¯1   0  0 ¯1
¯1  0  1   0  0  1
```

J6A was a set of planes from J6-0046, and we have computed the equivalent indices for these planes in the system used to describe J21-816. (It should be noted that X-ray diffraction cannot distinguish between 0 2 1 and 0 2 ¯1).

Weiss Zone Law

A set of faces whose normals are coplanar defines a **zone**, and the normal to the plane so defined is the **zone axis**.

Let *Z* [u v w] be a zone axis and *F* (**h k l**) be the Miller indices of a face in the zone. Transform both *Z* and *F* to Cartesian by multiplying by *D* (direct lattice) and *R* (reciprocal lattice) respectively. Because *Z* and *F* are orthogonal,

```
      (Z+.×D)+.×(F+.×R)
0
```

F is a vector, *R* is ⍉⌹*D*, and multiplication of matrices is associative; consequently this identity can be rearranged:

```
      Z+.×(D+.×⌹D)+.×F
0
```

Hence the condition that *F* lies in the zone *Z* is

```
      Z+.×F
0
```

This is the Weiss Zone Law. The derivation given is much simpler than any other that I have seen. Indeed, a student accustomed to using *D* and *R* to convert from crystallographic coordinate systems into Cartesian finds the Weiss Zone Law almost trivial. I give the example in order to emphasize that experience of working in APL not only gives numerical results, but provides important insight. The consistency of the notation and the ease with which matrices are used make APL invaluable in teaching elementary crystallography.

Conclusion

In the review of a mineralogy textbook, Derek Flinn (**Geological Journal**, vol. 8, April 1972, p v-vi) said:

> "I believe that the mystical approach to Miller Indices should be abandoned and that the student should be told that they are face-normal vectors, that they belong to the same vector space as the zone axis indices and that the addition and multiplication of these indices are elementary operations in vector algebra."

Flinn is correct, and I can testify that APL makes what he recommends easy and natural.

TIME-SHARING INFORMATION MANAGEMENT SERVICE

Roger D. Wilson
,Xerox Corporation
Webster, New York

I would like to introduce you to the conversion of Xerox Time-sharing processing from I.P. Sharp Associates in Toronto to an in-house system through a review of a business proposal which was written in October of 1975 by Chuck Swedrock, Norm Schueckler and Ken Gannoe. In this business proposal, the first issue covered was background of time-sharing processing at Xerox.

In early 1972, opportunities were identified that justified APL time-sharing on Sigma computers in the Xerox Computer Center (XCC) in Webster, New York. Since that time the ongoing emphasis has been to provide cost effective alternatives to the purchase of these services from outside vendors.

Time-sharing Service at XCC Webster began in January of 1973. Although there were start-up problems in those initial systems, the internal chargeback revenue had grown from $40,000 per month in the first quarter of 1973 to a 1975 level of over $200,000 per month. This growth in view of difficulties in the internal operation is an indicator of the persistent emphasis on this form of information services development. Therefore the primary issue presented in this proposal was the alternatives for supporting that forecasted growth in time-sharing applications in 1976 and beyond.

At the time of this proposal, Time-sharing Services for Xerox U.S. Operations costing in excess of $180,000 per month were being purchased from external vendors. This outside expense was conservatively forecasted to grow to over $260,000 per month in 1976 unless action was taken to provide increased capability from internal computer facilities at XCC Webster. With the present internal time-sharing operation based on Sigma computers nearly saturated, an alternative emerged with potential to achieve major net expense reductions in 1976 and beyond by transferring existing and planned time-sharing applications to an in-house facility. The recommended strategy included acquisition of an IBM 370/158 computer in 1976. Time-sharing technology was to be provided without major internal development efforts by purchasing software and consulting services from I.P. Sharp Associates. The advantages of this approach as compared to any other alternative are summarized in three major areas: first, increased user confidence with the in-house facility due to the technical compatibility with consultant services obtained from major external vendors that were currently supplying the service. Second, increased internal control over Corporate sensitive data now handled by external vendors, and third, an opportunity to implement new cost efficient data storage technologies.

Let's take a look at the use of time-sharing within Xerox and how it has grown. The proposal showed that XCC and vendor supplied time-sharing in Xerox increased from $20,000 per month of usage in January of 1972 to over $400,000 per month of usage in July of 1975. There were two major questions to be answered in the initial proposal. First, would time-sharing growth continue, and second, at what rate? First it is necessary to recognize the profile of time-sharing usage within Xerox. Literature commonly assumes that time-sharing is used as an interactive problem-solving tool, and in most cases outside of Xerox this is probably a valid assumption. However, within Xerox much the opposite was, and still is, true and we find that time-sharing is used as an integral part of the business management process, and thus is likely to continue its growth because of the pressures to reduce cost and improve operational efficiency.

Certainly a critical issue in time-sharing is the outlook for continued growth in the face of difficult management problems with respect to current business pressures. Justification for incremental spending will be especially difficult when many areas of our company are concentrating on the reduction of overhead expense. In a deeper look at time-sharing as a business tool we find historically that the major time-sharing applications were developed as a series of modelling techniques to assist in planning and forecasting activities. Used in this way, these systems could have remained minor interactive exercises typical of the classical or conventional mold. However, as the models were used it became evident that time-sharing APL provided a powerful responsive control and communication tool which was integrated into day-to-day operation and control.

Again, an expected but dramatic change in time-sharing technology provided the impetus to demand. Massive multi-megabyte files became not only possible but economically practical. Thus, a model of modest proportion could be expanded to produce a complete operating plan with the further ability to add both actual data and current-outlook data to provide all levels of management with information in time to creatively react to a changing business environment. In a major Xerox application as product volume, variety, manpower, expenses, customer installations and revenue continued to grow, the volume of data storage and time-sharing usage grew proportionately. Again it is imperative that we recognize the real impact of connecting this concept of massive data files to a communication network based on time-sharing systems. This combination provided a high level of responsive information integrity at a reduced overhead cost. In fact, Xerox Region offices reported reduced overhead manpower costs, elimination of paper-filled cabinets and the capability to respond to the increased pressures of competition with accurate and relative information.

This same growth pattern was experienced in Logistics and Distribution control procedure. In this case the major driving force behind the growth rate, in addition to the use of a reliable file oriented time-sharing service, was the as-needed availability of customer support or consulting services personnel at all local or remote locations. This support service was and is a major factor in the continued success of these systems which operate from many remote locations.

Today, time-sharing is crucial to Xerox; it is crucial to all levels of management for informed decision-making; it is crucial to control by providing current factual information, and it is crucial to communication between all levels of control for every facet of activity comparison to plan. Further, it becomes more crucial every day as business, economic and competitive pressures increase the demand for cost-effective information communication at all levels of management. Thus, under the constraints of a reduced overhead environment, it is reasonale to forecast continued growth in a cost-effective data-file-driven communication-network-based time-sharing service. Time-shared APL has demonstrated its capability as a viable control and communication method of operation for management in a competitive market. Attempts had been made, prior to the time of this proposal, to convert the workload running in Toronto to the Sigma-based systems at XCC Webster. Those attempts did not prove successful primarily because of problems related to the hardware and to the power supply at the computer center.

The general direction of this proposal was toward an IBM based system which would support the SHARP APL software. The major advantage was that this strategy was based on the fact that two-thirds of the workload at that time was on IBM equipment at outside vendors. The objectives satisfied by this strategy are therefore minimum software development, minimum project duration, minimum training for Xerox personnel by vendors, and a minimum of exposure for disruption of the time-sharing user.

The basic steps of the installation called for the placement of a 370/158 processor configuration at XCC Webster in 1976. Step two was the installation by I.P. Sharp Associates of their operating system, an APL processor, and in addition, the training of Xerox personnel by I.P. Sharp and Sharp's involvement in the conversion of APL systems from the system in Toronto to our system in Webster. I.P. Sharp also committed to the continued maintenance and support of their APL processor running on the Xerox system. Use of software and consulting services from I.P. Sharp Associates was viewed as a requirement for this strategy in order to provide the shortest project timetable with the smallest risk of conversion failure, and also to eliminate the necessity of a long development project by Xerox to provide necessary software facilities.

After numerous staff and management reviews this project was given the green light for implementation. From here on, the project became a two-pronged effort. Our first area of concern and effort was with the users of the system. In order to come at the concern and questions of these users who were historically very satisfied and content to operate in the existing Sharp environment we set up a user review board. This user review board was made up of people from the systems environment as well as users from the functional areas. We selected people whom we considered to be in key positions of both decision-making and actual experience to be part of this review board. In regularly scheduled meetings and through the presentation of formal documents, we gave to these users a scenario of the actions that were to be taken during the project planning and the transfer of the workload from Toronto to Webster. We stated to these users the level of service that we felt would be provided and the type of support that we felt they needed and we would be able to give them. Many questions were raised by the Review Board, all of which were answered carefully and to the best of our ability.

In order to enhance our own knowledge of the operating environment from the user point of view, people on our project staff visited many of the user sites and talked to the people who were actually at the terminal and interfacing with the system. When the system was initially installed we identified certain members of the Review Board who we considered to be amiable, and if you will, on our side, to become test users of the environment for a period of approximately one month. Because we were asking them to act as guinea pigs there were no charges from our processing center to them as users during this trial period.

The second prong of our effort was in the direction of the actual implementation of the processing environment.

In our many discussions and negotiations with I.P. Sharp Associates, we had determined that there were key elements of this operation which had to be injected into our own personnel, both operator and support personnel. A period of training for our operators and software technicians was worked out with I.P. Sharp Associates. This training included formal classes held for Xerox personnel, simulated operations at Toronto and hands-on experience with our own equipment, guided by a Sharp operator located temporarily in Webster, during the time when our hardware was in operation, but prior to load transfer. In addition to training, we worked out with Sharp a profile for user workload and the methodology for transferring that workload from Toronto to Webster. Telecommunication requirements were identified and solutions developed. Our final solution included a combination of Xerox private data network and use of the I.P. Sharp network within the United States. In order to have all users transferred from the Toronto operation to our Webster operation, it was necesary for us to implement a blocking mechanism at I.P. Sharp Associates so that users who logged on to that environment would be told that their files had been moved to Webster. In addition, we made trial runs to unload files and workspaces at Toronto and to reload those files and workspaces at Xerox in Webster, and asked users to come on to the system and test in that new environment.

The final transfer of workload was made on the weekend of July 19th, 1976. At that time, all files and workspaces of Xerox users in Toronto were unloaded in Toronto and reloaded on our system in Webster. Users began operation the following Monday and although we had provided for the possibility of having to move the files and workspaces back to Toronto in case of a major disaster in the early time of our operation in Webster, we have never had to exercise that capability.

During the past two years we have taken many steps to improve and upgrade the processing environment that we supply to our users. We have implemented a customer information center to handle calls from users and provide a solution or to hand off that call to a technician if necessary. In addition, through use of hardware and software measurement and monitoring equipment, together with Sharp technical personnel we have made numerous enhancements to our environment. These enhancements have allowed us to continue operation on a 370/158 even though the workload has grown significantly larger than was initially projected for that hardware. Our CPU has grown from a 158 model 1 to a 158 model 3. This swap of CPU's occurred in October 1977, about 15 months after load transfer, with only minimal impact, about two hours of unscheduled down time, to our users. Service from XCC-Webster currently handles users from Xerox locations across the United States, Mexico City, Rio de Janeiro, Brazil, and limited use from London, England.

At this time, we are in the process of developing a proposal to expand the capacity of our SHARP APL system to provide for growth in the U.S. user base and also to provide processing for Rank Xerox users in the U.K. and Europe.

In conclusion, we have found the move from use of outside time-sharing service to an in-house service to have been a very worthwhile effort. It has provided us not only with significant financial advantage but also with the ability to more carefully monitor the growth and increased requirement for APL services in Xerox.

MIGRATION TO AN IN-HOUSE SYSTEM

Maurice Elliott
Massey Ferguson Limited
Toronto, Ontario

Abstract

I want to begin my talk today by telling you a little about Massey Ferguson and the way we are organized. I will describe our use of APL, and the way it has spread since we first experimented with it over six years ago. Then I'm going to outline the sequence of events which led to our justifying the running of our APL work in-house and made it feasible for us to do so. Then I'll describe how we planned and organized the move in collaboration with the I.P. Sharp staff, and the way it worked out in practice. Finally, I will discuss the level of service we are getting, and the reactions of our users to the move.

Introduction

Massey Ferguson is a fairly large multi-national corporation based in Toronto, with annual sales of around $3 billion. We have some 90 factories in 50 countries throughout the free world. Our products are sold in almost every country of the world, except for the planned economy countries. Our major product is, of course, farm machinery. We also have substantial sales in the industrial and construction machinery business. The Perkins Engine Group, one of the world's largest manufacturers of diesel engines, is a wholly-owned subsidiary, and contributes the third major slice of our sales.

Massey Ferguson is organized at three main levels. At the first level, we have Operations Units (or OU's). An OU is based in a country where we have factories. It sells its products in its own home market, to other OU's for sale in their home markets, and to export countries where there is no factory. Regions form the second level of organization. We have split the world into three regions - Europe, the Americas, and the rest which we call A/A/A (for Africa, Asia and Australia). The regional staff are responsible for coordinating the activities between their operations units and markets. At the third level, we have the Corporate head office. Because we have a strong decentralization policy, the Corporate office is quite small and relies heavily on the regions and OU's for the short term planning and coordination. The Perkins Group can be thought of as a fourth region, though it is totally distinct from the MF structure. The reason for this is that the Perkins Group has business associations with many of MF's competitors who would naturally be very upset if their plans were passed on by Perkins to MF.

It's interesting to contrast the characteristics of the three MF regions. They are quite different, and it leads to some interesting challenges when we try to move a system from one location to another, or to write a common system. In the Americas region, we have North America - a mature, politically stable market, and Latin America - much more volatile, both politically and economically. These two parts of the region have never been really united, so vast are their differences. But on the whole, the OU's are relatively self-contained. There is little import or export trade. The OU's in Europe, however, trade extensively among themselves, and export a great deal of their production, both within Europe and to markets in the A/A/A region. European systems therefore tend to concentrate on managing production facilities efficiently, particularly in the multi-currency environment. The A/A/A region, on the other hand, has only two small OU's. Most of the goods it sells are imported, mainly from Europe. It also coordinates activities in countries such as India and Turkey, where our products are assembled locally from imported kits of parts. The A/A/A markets cover the entire spectrum from primitive to mature, politically explosive to very stable. Its systems therefore tend to concentrate on managing the mass of data needed to plan its sales and marketing strategy in the face of these problems.

Behind all this, the Corporate office has the responsibility to coordinate the activities of the OU's and regions. It does not become involved in day-to-day operations, except where decisions such as inter-company transfer pricing affect more than one OU or region. Instead, it concentrates on the future, on the alternative scenarios that might develop, and on answering the "what-if" type planning questions. Thus it was an ideal spawning-ground for APL at MF.

The Growth of APL Use

We were first introduced to APL in March 1972 by two I.P. Sharp Associates (IPSA) analysts who had just programmed a batch system for us in COBOL. They took a copy of our data base, and demonstrated what could be done with it in APL.

For over a year we made little use of APL, but after we installed a couple of systems very successfully, its use began to grow. At first we used it only for Corporate systems. When I.P. Sharp opened their transatlantic line to London, however, we began to show European management some of the systems and facilities of APL we felt might be useful to them. Reactions to our efforts were interesting, to say the least. The reactions of MIS staff varied from mild interest to strong resistance. With their traditional batch-oriented background, they abhorred the way we appeared to install systems too hastily, with virtually none of the paraphernalia of analysis and system design to which they were accustomed. The users' reactions, on the other hand, varied from mild interest to wild enthusiasm. Indeed this enthusiasm, and our ability to satisfy it in many cases where the local MIS group did not have the resources or tools to do so, has on occasion caused friction between the Corporate and local MIS groups, despite all our attempts to prevent it.

Another cause of problems was the quality of the communication between our European locations and the IPSA machine in Toronto. In the early days it was quite common to get a line that was so bad you were unable to print a page of output without getting several scrambled lines. At the time, we were looking at ways to increase the capacity and reduce the cost of our telex network between various locations. We found that we could install a private telex network which gave us much greater capacity, cost much less, and would also handle our APL load. We proceeded to install this network, and gradually provided links to the Sharp machine from our centres of APL use. Most of our European users now have the luxury of hard-wired terminals which can be, and are, expected to behave flawlessly for hours at a time.

As all this was going on, our use of APL was growing rapidly, along with our bill. In 1977, despite generous discounts, our APL bill was about $1.5 million, and we expected it to be around $1.8 million in 1978. So at the beginning of 1977 we started looking for ways to reduce the cost. The obvious possibility was to run SHARP APL in-house. But this would have required us to justify a separate computer to run it since all our existing computers ran under virtual operating systems. In view of this, we decided to take a fresh look at the field to see if we could find a suitable alternative to SHARP APL.

The first contender was, of course, IBM. They were quick to tell us how they were going to support and promote APL in the future. But their APL lacks many of the features of SHARP APL (a good file system and report formatting, to name but two), and they of course could not give us any solid assurances that these deficiencies would disappear. The little that we did get in the way of promises for the future from them was certainly not a sufficient foundation for us to decide in their favour. We also looked at the then new APL on the Hewlett Packard HP3000 minicomputer. This we quickly rejected when benchmarks showed it ran 30 to 50 times slower than SHARP APL. The third major contender, which came in strong to the end, was the APL product offered by DEC for the Dec System 20. Benchmarks showed its speed was quite adequate, and it had some features which we felt were superior to SHARP APL (restartability, for example). On balance, we considered it to be comparable to SHARP APL for our needs. But despite the fact that the DEC APL had facilities equivalent to almost all those in SHARP APL, we realized that the conversion effort would be enormous. The problem was the number of workspaces and systems we have, and the interdependencies between them. If we were to justify the hardware cost and drain on people to convert to, and run DEC APL, it would have to be because the outcome would be significantly better, and the payoff very quick. This we certainly could not commit ourselves to, and so we also rejected that option.

The result of this exercise was that we renewed our commitment to SHARP APL, but searched for ways to make it cheaper. At the same time, gloomy financial forecasts for 1978 were getting gloomier (and unfortunately more accurate). We realized that we had to find some way to reduce the increasingly visible (and commented-upon) cost of APL. Of course, in that environment, all our operations units were similarly looking for ways to reduce costs. And one of the ideas that we developed with North America was to consolidate the two main data centres in Des Moines and Toronto into one, at Des Moines. This simultaneously gave them substantial savings and a big headache. The data centre at Toronto would become a remote job entry station. The 370/158 there, together with about half the space, would be redundant. Moreover, the 158 was purchased and to sell it would involve a substantial loss on the book value.

This gave us the opportunity to justify a joint project to consolidate the North American data centres in Des Moines, and bring APL in-house on the Toronto machine. Since we both wanted to achieve the maximum possible savings in 1978, we set ourselves ambitious project schedules. North America planned to move their load at the beginning of March. Assuming we would be able to get dedicated time on the 158, we then scheduled ourselves to begin in-house operations around the middle of April, after six weeks or so of testing.

Transfer Plans

Our plans for the move involved several aspects - hardware and facilities; the operating system and SHARP APL software; our applications software and programs; telecommunications; user communications; and last, but by no means least, staffing. I'm going to talk separately about the plans we made in each of these areas. But first, a general comment. Peter Teeson became our main contact at IPSA for this project, and we held weekly planning meetings which he attended. Other I.P. Sharp staff members came when it was appropriate. Their many words of wisdom and caution (prophecies of doom, in some cases were greatly appreciated, and our success was due in no small part to the effort they all contributed to this project.

On the hardware side, our immediate action was to educate ourselves about what would be needed to run APL. We took an inventory of the hardware in the data centre and found that about half the peripherals and memory were redundant. We gradually found buyers for these and sold them. The three main problems were the 3705 communications controller, the card reader, and the printers.

The configuration of the 3705 was totally incompatible with the communications software Roger Moore wrote for the SHARP APL network. We promptly ordered a suitable 3705 from IBM, but the delivery time we were quoted was the fall of 1978, and despite the pressure we put on IBM, they would not promise a machine in time. For several weeks this issue dominated the hardware side of our planning meetings as we considered alternative strategies. Eventually we got IBM to agree to strip and rebuild our 3705 to a more suitable configuration. They could not give us everything we wanted, but with an Alpha concentrator we could get by.

The second problem was to determine whether the existing 3505/3525 card reader/punch could be run under the Sharp operating system (a stripped-down DOS). Opinion was that it could, but if not, we would have to get a compatible reader such as the 2540. There really was no way to check this out except to try it, which we finally did, and it worked with a few changes to the error recovery routines.

The third problem area was system printers. There were two impact printers in the data centre, either of which would have served us well except that they did not have APL print trains. However, there was also one of the new IBM 3800 laser printers which would be retained. This can print five pages a second, using any graphics you want provided you define them. We found that IBM does not offer an APL character set for the 3800, but we managed to get one from Stamford University, and have now produced our own 15-pitch version.

The 3800 gave us another problem in that it was acquired by North America to handle the print load supporting the Canadian factories. Of course, the planned move did not change the print load. So they would either have to move the 3800 printer to Des Moines, or do the printing remotely in Toronto. It was not practical to move the printer because of the volume of paper that would have to be shipped. On the other hand, those of you working in communications will appreciate the problem of feeding an ordinary impact printer remotely, far less one like the 3800. We finally solved the problem with a communications box made by Comten. This box supports multiple parallel communication lines between locations. Not only does it use the aggregate bandwidth of the lines, but it also degrades gracefully if one or more of the lines goes down.

But the 3800 was still not finished with us! We had no way of feeding it directly from the RJE terminal at the Toronto end, so we had to find a way to off-line print the tapes spooled from the terminal. At that time, IBM had not shipped an off-line print feature for the 3800. So we had to connect the 3800 to the 158. I suspect we are the only IBM location running a 3800 under DOS (DOS thinks it is a 1403 printer!). Kerry Liles, who became our software specialist, wrote a spooling program which runs in the background partition on the 158 and prints tapes on the 3800.

On the data centre facilities side, we were helped by the fact that the data centre already existed and was generally adequate. We moved the CE room, and took over the old one for the bridge, which we equipped basically the same as the Sharp bridge. This was the only major change in facilities.

The APL and operating system software side of our move was much easier to plan than the hardware side. Our first action was to negotiate a contract with IPSA to use their APL software, and for their support to install and run it. I.P. Sharp gave us a list of the software we would need, and how we should get it. In many cases, getting the software from the official source was merely a formality (except for the cost) because IPSA had changed (i.e., fixed) the version they ran, and gave us the changed version. The software plans in fact were the simplest aspect of our total plan. We anticipated, and found, few problems in this area when we finally did the transfer.

Our systems and applications software presented somewhat more of a problem. Not the least of our concerns was the sheer volume of stuff to be transferred - some 500 accounts, 1500 workspaces, and 450 megabytes of files. Our target was to transfer the work with the very minimum of interference with, or risk to, any development or production work. We quickly made three decisions which reduced our workload to manageable size. The first decision was to not attempt to move systems one-by-one. We had too many interdependencies between systems to do that. The second decision, which was forced by the first one, was to transfer everything twice. The first time would give us some experience with the problems we could expect; and between the two transfers we could run our system in parallel in test mode, leaving all production work safely on the IPSA machine. The third decision was to create on our own system the same account numbers we had been using at IPSA. In one fell swoop, this decision removed dozens of problem areas such as file names, access matrices, hard-wired account number checks, etc. We were also fortunate in that we were able to preserve the sign-on locks on all accounts. To do this we did not break security, but transferred them in their internal encoded form.

One major security problem, over which we lost a lot of sleep, was the question of locked functions. For obvious reasons, IPSA would not allow us to walk out of their data centre with dump tapes containing locked functions until they were very sure we were entitled to take them. IPSA's first line on this was that they would not let us take **any** locked functions. System software (HSPRINT, BGSORT, etc.) would be transferred separately and we would have to fix the rest ourselves. We complained long and bitterly about this attitude and showed IPSA how enormous our task would be unless they bent a little. Including all repetitions, we had about 5,000 occurrences of locked functions in our workspaces and files, and knew we would be entitled to almost all of them. Moreover, in some cases we did not have unlocked versions of some of our own functions anymore. Finally we agreed on a compromise. We assigned every locked function to one of the following groups:

1. system facilities (HSP, etc.)
2. proprietary software packages (e.g., Mailbox)
3. other locked functions from public libraries (e.g., statistical functions)
4. functions whose names correspond to locked functions in non-Massey users' accounts
5. functions whose names are unique to our accounts

Since we were buying the system facilities as part of SHARP APL, we would have the functions in group 1. They would be different functions because the system file passnumbers on our system are different. Where we paid for the packages in group 2, IPSA staff installed them on our system. Similarly, in group 3, workspaces we paid for were transferred. Group 4 presented us with our only real headaches. Fortunately there were very few functions in it. In each case, we had to convince IPSA the function was ours before we could transfer it. Group 5, of course, was no problem.

Planning for the transfer of locked functions occupied a significant part of our efforts. But during the transfers, our planning paid off and there was only a handful of problems with them.

Telecommunications gave us some work, too. When we started the project, we had a network serving some of our European locations and a transatlantic line connected to the I.P. Sharp computer. By the end of the transfer, we had to expand it to cover all our European factories, install seven new Alpha concentrators, and move the Toronto end to our data centre. We already had a plan for the expansion of our network in Europe, but we now had to shorten the time-scale drastically. Nevertheless, the expansion went well and was completed in time.

The difficult task was to re-route the Toronto end of the line. In addition to APL traffic, our transatlantic line also carries all our telex messages between the European message-switching computer (in England) and its North American counterpart (in Toronto). At all times, we had to avoid any lengthy disruptions of either service. We agreed to schedule all network changes either several weeks before we installed APL, or to postpone them until several weeks afterward. The I.P. Sharp consultants were not interested in trying to simultaneously debug a new APL installation and the network. Again, those of you involved in communications will recognize the wisdom of this advice. Since we wanted to wash our hands of the project as soon as possible, we decided to schedule the move before the APL installation. We knew we could change our plan in this area easily because either way the network could access our in-house system. In fact we had a few problems and almost had to reschedule it, but managed to squeeze it in just two weeks before the installation.

We also had to get telephone equipment, racks, an Alpha concentrator and other equipment installed in the data centre. All of this was straightforward, and went without a hitch.

User communication was a fascinating question. Should we describe our plans in detail to our users and, for example, warn them of the problems we could have; or should we tell them the bare minimum and hope we didn't have significant problems? In fact, we had four audiences for this question - senior management, Corporate analysts, analysts at other locations, and end users. To our senior management we tried to give a balanced view of the benefits and risks, though not in great detail. Corporate analysts were in close touch with the project, and could be expected to tolerate a few problems and pitch in to help if necessary. The other analysts were kept fully informed and involved in our plans. This gave them the opportunity to comment, make suggestions, and commit themselves to its success. While they were willing to help, there was little they could do except to make sure the network was working. To the end users we made several presentations. We told them what we were doing, that there would be a slight change to their sign-on procedure, that their bill would be smaller, and that they should see no other changes. At the same time, we surveyed all users and analysts to find what public workspaces they were using so we could negotiate with IPSA for them. Analysts in the region were asked to coordinate with their users to raise any questions or problems they might have.

Finally, our staffing plans. The price we paid for SHARP APL included technical advice and support to install it, and an IPSA operator for three months to help us run it and train our operators. We had to find the permanent staff for our data centre. At first we thought we could take over and train the operators who were being made redundant as a result of the consolidation of the North American data centres. But most of them were not interested in moving from an MVS shop to a DOS shop, and most of the others were not acceptable to us. So in the end we only transferred one operator and the operations supervisor from North America. We hired two community college graduates to complete our operating staff. Since we planned to run our data centre the same hours as IPSA, we were running very tight on operating cover, as the IPSA staff pointed out. For economy's sake we chose this course of action, realizing we might have to fill in with an operator hired temporarily from IPSA, run APL unattended at off-peak hours, or simply shut it down when there is no usage.

In addition to the operations staff, we needed a software analyst, communication specialist, and data centre manager. Kerry Liles, a senior systems analyst at our Corporate office, expressed interest in the software position. Since he had worked for IPSA and spent some time in the "Zoo", we took up his offer. We also promoted two others from our Corporate staff to fill the other positions. One further position, to which we promoted our secretary, was that of system steward. She is responsible for coordinating billing, enrolling new users, controlling access to other systems such as the Mailbox, and generally tying together loose ends.

The Transfer Itself

After all that planning, we might have expected the transfer to go smoothly, but we had too much respect for Murphy's Law. Our basic timetable was to have the hardware and software up and running by mid-March - two weeks after North America handed over the data centre. We would then do the first transfer of all our workspaces and files from IPSA. For the next six weeks, we would run the system in test mode, training our operators, testing the network, and ironing out any bugs. In the meantime, all our production work would be done on the IPSA machine. Finally, if all went well, on May 1st we would again transfer everything, and leave the IPSA system.

The first setback was in the North American schedule, which was delayed two weeks. Further problems pushed the final hand-over back to the end of March though we did get time on the machine before then. Because we had been rearranging the data centre since the middle of February and running all the testing we could do before we got the machine to ourselves, we were ready to begin testing APL and the 3800 spooling program as soon as the hardware was ready. On Friday March 17th we dumped all our workspaces and files at IPSA (onto 12 reels of tape!) and by Sunday we were ready to load them on our machine. Then we found that the restore of a selective dump of a large number of files was **very** slow, because it was testing for the presence of each file before loading it. The last straw came when one of the tapes broke. So back to IPSA we went to dump our files again. This time, having taken the test out of the restore program (and copied the tapes!), we loaded the files without a hitch.

Before taking the dumps, we had prepared decks of "pass" cards to allow our own locked functions through onto the dump tape. The dump programs listed the functions they had not passed, and we had to fix these where we could. For the first transfer, we only fixed workspaces (the majority of the problem, anyway). From a privileged terminal we loaded each workspace, copied in the erased functions, and saved it. This was done very quickly using the programmable keys on an HP2641 terminal.

Finally we set up the account lock table by transferring the encoded locks from the IPSA system.

By this time (March 23rd) we were getting very short of time to adequately train our staff for production running and test our applications. Our target was only five weeks away, and even our original plan of six weeks testing was regarded as too short by the IPSA consultants. We now had to decide whether to push for our original target, or to postpone the final transfer. One fact which weighed heavily in our decision was that on May 10th we would begin using APL to process and analyze our world-wide financial results for the second quarter. Since we could not jeopardize that process, we could not transfer on the weekends of May 6th or 13th. In addition, there was a significant cost impact if we postponed. So rather than delay the transfer for three weeks, we decided to push for May 1st. Of course, we could have changed our minds up to the last minute if we ran into serious difficulties.

In the meantime, we were busy training our staff, and testing our applications. We checked that we could access our system from all locations through our own network. We also set up the Mailbox and other IPSA packages on our machine. During this period, our testing went well. The communications network settled down remarkably quickly after the changes. So we decided to continue pushing for the May 1st target.

For our final transfer, we arranged to have all our accounts locked out of the IPSA system at 1 pm on Friday, and to complete the transfer over the weekend. Despite the fact that the first transfer took over four days, we felt that we could do the second one twice, if we had to. Our reasoning was that we knew what to expect, already had APL up and running on our machine, and had a detailed schedule of activities to follow. The only changes from the first transfer were that we planned to re-file everyone's Mailbox messages, and to repair locked functions erased from packs in files, as well as workspaces.

At 1 pm on April 28th, we began the final transfer. Our accounts were locked out, and the encoded locks were dumped. The lock on each account enrolled in the Mailbox was changed. Then Kerry Liles signed on to each one and ran a function he had written to save all outgoing Mailbox messages in a file. We then dumped all our workspaces and files onto tapes, copied them, and restored them at our data centre. Kerry Liles signed on to each Mailbox account and ran another function which re-filed all the outgoing messages. Then, as before, he replaced the locked functions in workspaces which the dump utility had erased. This time he also ran a function which IPSA wrote to replace all locked functions that were erased from packs in files. Finally, the account lock table was restored as it had been on the IPSA system.

All this took until just after noon on Saturday, at which point most of the Corporate and European analysts descended upon the system to test it. We found a few problems. For example, we had a few instances where a locked function in a file component was originally held in a pack within a pack. The fix function, not expecting this, had replaced them in the outer pack. But the few problems we did find were very minor, so we breathed a sigh of relief and waited to see what Monday morning would bring.

After the Transfer

On Monday May 1st, our in-house system came up as though nothing unusual had happened. And it continued to do so throughout the month. We ran into a few cases of locked functions which had not been replaced because the dump utilities had not told us they had been erased. Each one was, of course, obvious and quickly fixed.

Throughout May we held our breath, waiting for the big problems. But there were none. In fact, our only unscheduled down-time happened late one Friday afternoon when a power transformer failed, cutting off power to the entire data centre for four hours. This reduced our up-time record to only 99.7 percent! We found our response time was good, though compute-bound jobs take longer than at IPSA because our CPU is slower than the Amdahl 470. Overall, however, our users were very satisfied with the move.

The month of June was very similar. An operator problem kept the system down for about four hours one Saturday. And on three occasions, the system crashed and was down for about fifteen minutes each time. We strongly suspect these crashes were caused by DOS failing to handle channel checks on the 3800 printer (which it thinks is a 1403!). This problem will disappear when IBM installs the off-line print feature for the 3800.

Summary

What did we achieve with this exercise and what did we learn from it? From the company's point of view, our most important achievement was that we reduced the cost of APL (and made it more or less fixed) without sacrificing the quality of service. We also demonstrated that with teamwork and careful planning, an in-house migration can be done quickly and without disrupting access to APL, even by naive users who simply know how to sign-on and run a particular system. Another important achievement, at least in the eyes of our management, is the increased security we now have with all our work being done in-house.

The most significant things we learnt from this exercise were the importance of our very close cooperation and involvement with the IPSA staff. We also gained new respect for Murphy's Law! As you can see, our migration was not made without some unexpected setbacks. But we had made detailed plans, left some slack for unexpected problems, deliberately considered alternative plans for tricky areas, and taken seriously the advice given by the IPSA consultants. Because of this, the surprises did not make us lose track of where we were and where we were going. The majority of our decisions had been thought through in advance, and we had no panics to solve quickly. This fact alone reflects the quality of the work which both companies put into the planning and execution of Massey's In-house Migration.

Apologia

At the time I agreed to give this talk, I was closely involved with the planning of the transfer. Later I had to withdraw from the team due to the workload of another project I was working on. Thus, except in the early stages of the planning, the term "we" I have used generally does not include myself.

SHARP APL AS A DISTRIBUTED PROGRAM PRODUCT

Richard H. Lathwell
I.P. Sharp Associates Limited
Toronto, Ontario

SHARP APL

Eric Iverson has described the evolution of SHARP APL ("The Commercializing of APL\360 and Some Plans for its Future"), elsewhere in these proceedings. In summary, I.P. Sharp Associates began offering a time-sharing service in 1969 using APL\360, then a Type III offering which had been developed in IBM Research primarily as a device to bring APL to classrooms for the experimental teaching of mathematics. Although this system had many advantages, it lacked many facilities essential to commercial data processing, the most obvious being access to large volumes of on-line data. IBM did not address this problem in an APL product until APLSV was announced in 1973, and in the interim a number of time-sharing vendors and software houses developed their own solutions.

The major factors which influence the evolution of SHARP APL are customer requirements. Attempts to address them lead to the development of enhancements and their introduction into the vigorous environment of a heavily used time-sharing system, a process often called a "smoke test", a term borrowed from hardware development and designating that part of the development process when power is first turned on. There has been occasional smoke and even a few fires, but SHARP APL continues to thrive and attract business, system usage continues to increase; customer acceptance is the ultimate measure of the success of development efforts.

The SHARP APL network has evolved in a similar manner. As time-sharing usage increased, users became increasingly dispersed geographically as they began to require access from all of their office locations, and multinational organizations desired international and intercontinental access. Early attempts to use commercial common-carrier-supplied data communication indicated that although long range communication could be achieved, it is slow, unreliable, and very expensive. Common carriers must provide general solutions to everyone's communication problems, and as a result they do not provide an optimal solution to any specific data-communication problem. A synergistic set of events, a lot of work, (and a lot of smoke as contemporary users will attest) led to the installation of the SHARP APL network: a private, packet-switching system constructed of minicomputers and leased highspeed data links, which interconnects the SHARP APL system components and users.

Distributed Processing

The APL world has doubled each year for the past several years, and the use of SHARP APL has grown apace. Growth has its problems, particularly when it must be supported by the installation of computers and their peripherals, by increased floor space requiring a controlled environment, and by increased support staff. SHARP APL became a distributed system much in the way that other major changes have occurred: through an attempt to meet a customer's requirement. In 1974, the use of SHARP APL by the Xerox Corporation had become significant, was exhibiting the growth typical of APL when used for data processing, and it was clear that sufficient hardware to meet Xerox needs would be difficult to install in Toronto. It was also apparent that Xerox expenditures for SHARP APL Service were sufficient to fund a computer installation on their premises in Rochester, New York. The system was installed in June 1976, connected to the network, and by mid-1978 its use had grown to the capacity of the computer.

The success of the Xerox installation confirmed the belief at Sharp that the most effective way for the system to grow at a rate sufficient to satisfy customer needs is to distribute processing to customer sites, rather than to double annually computer power in Toronto. This concept gradually became real and set a direction for development and marketing. In May 1978, I.P. Sharp Associates announced SHARP APL as a distributed program product, and during the same month Massey Ferguson Ltd. became the second major SHARP APL customer to move their users to a computer on their own premises.

The Xerox and Massey Ferguson installation experiences are each described in these proceedings: "Xerox Moves APL In-house", Roger Wilson, Xerox; and "Migration to an In-house System", Maurice Elliott, Massey Ferguson.

SHARP APL as a Program Product

The decision to provide SHARP APL as a distributed program product has become a major influence on the current evolution of the system, leading to a great deal of introspection within the Development Group as it attempts to determine and address the associated requirements and to define the product. The current development direction is somewhat ideal, and attempts to reconcile product distribution and maintenance requirements, plans for future evolution, and continual system improvement, all of which must be carried out without adversely affecting the operation of existing APL application systems. Some of the factors which are now influencing SHARP APL are:

- Interchangeability of APL programs and systems and interaction with the professional APL community: The IBM APL Language Manual and other relevent publications are being used as a definition of "standard" APL; IPSA staff are involved with organizations such as STAPL (ACM) in efforts to minimize dialects of APL and to arrive at common representations for APL variables, functions, and systems for interchange.

- Proprietary SHARP APL system enhancements: System features such as the file system, packages, and formatting primitives provide customers with cost-effective facilities unobtainable elsewhere. These facilities are themselves evolving; both to increase their effectiveness and efficiency, and to align them with standard APL as definitions permit.

- Customer host environments: Although the recommended SHARP APL installation consists of a hardware system dedicated to SHARP APL supported by SHARP DOS (a small multiprogramming system which has evolved from IBM DOS/360 and POWER), many customers either do not wish to dedicate a computer to SHARP APL, or have requirements for other host operating systems. As contract negotiations proceed, development of support for particular host environments proceeds concurrently.

- Installation flexibility and packaging: As the variety of SHARP APL hardware and software configurations increase, an installation package is being developed to ease the effort required to install the system, to incorporate support for new hardware devices and variations in host operating systems, to provide a base for future enhancements, and to simplify maintenance.

A description of the product is contained in the I.P. Sharp Associates' fact sheet, "SHARP APL as a Distributed Program Product".

Product Distribution

The installation of SHARP APL consists of a complete facility design, plans for system operation, installation and verification of the software, training of customer operating and support personnel, and if network access is desired, installation of network support hardware and software, connection to the network, and re-configuration of the network to recognize a new processing node.

The process begins several months before installation with planning meetings during which the implications and requirements are discussed in detail, and during which schedules are negotiated. When the installation date becomes firm, a member of the SHARP APL Development Group is appointed as the primary liaison between I.P. Sharp Associates and the customer, particular software requirements are determined, and work begins on the packaging of the system to be installed. Interaction with, and the training of, customer support personnel begins a few months before installation, a trial installation might be performed, and eventually the system is installed, tested, and attended by a Sharp staff member during the early period of operation. The effort and resources required for a customer installation are such that a maximum of four installations can be scheduled annually.

The actual product installed is the contemporary version of distributed SHARP APL - essentially the system currently operated as the SHARP APL time-sharing service in Toronto. System problems can be analyzed and maintenance can be performed at the customer site or from Toronto for those systems attached to the network. New releases of the system are distributed to existing customers semi-annually, and contain corrections for problems determined in earlier releases as well as language and system enhancements which have been introduced and tested on SHARP APL time-sharing service in Toronto.

Among the factors which must be considered by prospective SHARP APL installations are ancillary user services. The services provided by I.P. Sharp Associates to their time-sharing customers comprise the majority of the costs of operating the Toronto service, and include:

- Operations support: Continuous system attendance, response to user requests for manual tasks such as tape mounting and the distribution of printed output, and the capability to answer or refer questions from users.
- APL support: Technical advice and assistance to users with APL programming problems or questions, guidance and advice in the design and development of APL application systems.
- Public library workspaces and Sharp proprietary application systems are products in their own right, and public data bases must be obtained from their licensing organizations.

A customer must evaluate the requirements of a community of users who are familiar with an APL time-sharing system, and must provide and train people to assist them during the transition and to meet their ongoing requirements.

SHARP APL as a Distributed Program Product

SHARP APL will continue to evolve as it has in the past. Research and development now underway in a number of areas will lead to the introduction of major enhancements over the next two years. Experience gained through the installation, maintenance, and support of a distributed system will lead to the recognition of new requirements and efforts to satisfy them. Ideas, suggestions, and criticism will come from interaction with the professional APL community, and inventions in APL, software, and hardware technology will become available. The intersection of all of these factors with the talent, resources, and interests of the SHARP APL Development Group will provide a semi-annual package of surprises for the users of SHARP APL.

CASH FLOW PLANNING SYSTEM FOR AN INTEGRATED OIL COMPANY IN A DEVELOPING COUNTRY

Karl M. Wiig
Arthur D. Little, Inc.
Cambridge, Massachusetts

Abstract

An APL based decision support system was implemented to project the required ten year future cash flows for an overseas integrated oil company. Included in the system are supports for (1) detailed regional demand projections; (2) transfer payments between subsidiaries; (3) investment requirements and repayment schedules for all manufacturing and distribution system expansion projects; and (4) consolidation calculations from subsidiaries to the parent corporation.

Introduction

The cash flows planning system described in this paper was developed for a medium-sized national oil company which is integrated from exploration and development, refining, distribution and transportation, through marketing. The oil company operates several refineries which are operated by separate operating subsidiaries. There are several marketing subsidiaries covering different geographic areas, and there are several other wholly or partially owned subsidiaries for transportation, lube and asphalt manufacturing and other activities.

For planning purposes, the demands are forecast for 30 geographical demand regions for each of 27 products over a 13-year planning horizon. These demands are redistributed to sales volumes at 40 bulk plants for calculation of sales revenues. Transfer prices and volumes are established for 21 transfer conditions between subsidiaries and between the outside world and subsidiaries.

Overview of the Cash Flow Planning System

The planning system which is described in this paper is developed for a developing nation petroleum company and is a simple computer-based system designed to support parts of the long-range planning process. In particular, it is designed to support the development of cash flow projections for the company as a total corporation for a period of ten years into the future (excluding the present year, for a total of 11 years). The cash flows are developed, maintained and reported on a subsidiary-by-subsidiary basis for the time period and then in the end they are consolidated into the company's total cash flows.

The planning system is specifically assembled to allow for the separate estimation, data maintenance and reporting of a number of different categories of data:

1. Annual energy demands for each of 30 demand regions by fuel type and resulting from either:
    - growth of known demands,
    - absolutely specified demands,
    - specific demands associated with any isolated projects across the country.

2. Sales **volumes** at each bulk plant (resulting from a redistribution of the above energy demands).

3. Sales **prices** at each bulk plant (together with the sales volumes, these generate the bulk plant sales revenues).

4. Transfer volumes of products and crude oils and transfer prices between subsidiaries (to generate transfer payments).

5. Imports and exports of products and crude oils between subsidiaries and external parties (to generate export revenues and costs of imports).

6. Markups and barter agreements for external transactions.

7. Cash flow payment schedules, start-up schedules, funding status, descriptions, manpower requirements and capacities associated with all of the company's projects (and those of its subsidiaries).

8. Annual cash flows not otherwise generated for all of the company's subsidiaries.

The approach for the formal (and computer-based) segments of the planning system has been to automate all the repeated calculations (such as projection of demands, multiplication of sales volumes by prices to produce revenues, and the posting of cash flow items into the cash flow statements). Extensive attempts have not been made to automate special calculations (such as estimation of a particular demand sector activity in a given geographic region or the assessment and development of the cash flows of a subsidiary as relates to its traditional operation except for sales and purchases of products). Instead, a computer-based system has been created to assemble the results of these manual analyses; and as the information changes, to update the data in the files of the computer by manual entries, and to report on the contents of these files as required for general information or for specific purposes.

The major objectives of the computer-based part of the planning system can be stated as follows:

1. Provide means to develop and store the basic data for projection of subsidiary and consolidated cash flows.

2. Provide facilities to edit and report on the basic data bases.

3. Provide means to assemble, edit, report and update the data on existing, planned and potential projects such as refineries, bulk plants, pipelines, etc.

4. Facilitate the calculations to produce subsidiary and consolidated cash flow reports.

The planning functions that may be considered to be required to project the company's and its subsidiaries cash flows are shown in Figure 1. Those functions that are included in the computer system are indicated as solid line rectangles, and those functions which are excluded are indicated as broken line rectangles. It may be seen that the energy demand projections to a large extent will require manual analysis (or at least will require analysis which is outside the present computer-based system). As the projections for demands change throughout the country, the required sizes of bulk plants, refineries and transport links will also change.

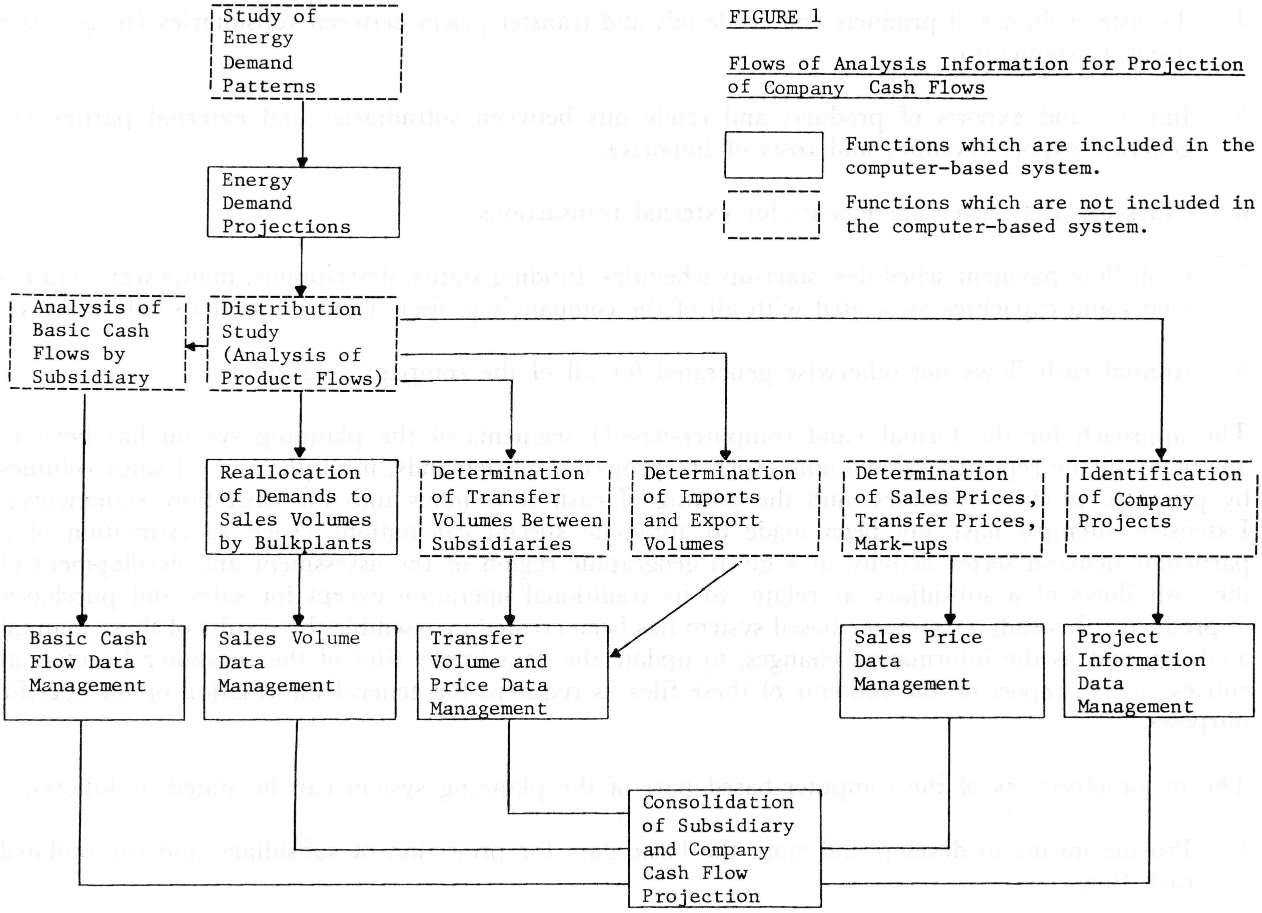


FIGURE 1

Flows of Analysis Information for Projection of Company Cash Flows

Functions which are included in the computer-based system.
Functions which are not included in the computer-based system.
Study of Energy Demand Patterns
Energy Demand Projections
Analysis of Basic Cash Flows by Subsidiary
Distribution Study (Analysis of Product Flows)
Reallocation of Demands to Sales Volumes by Bulkplants
Determination of Transfer Volumes Between Subsidiaries
Determination of Imports and Export Volumes
Determination of Sales Prices, Transfer Prices, Mark-ups
Identification of Company Projects
Basic Cash Flow Data Management
Sales Volume Data Management
Transfer Volume and Price Data Management
Sales Price Data Management
Project Information Data Management
Consolidation of Subsidiary and Company Cash Flow Projection

An analysis of the feasible and desirable product flows (including imports, exports and inter-subsidiary transfers) must be performed. From these results, the transport costs may be developed for inclusion in the cash flows. This distribution study will also provide indications and even direct specifications of the refinery, bulk plant, and distribution system projects which are required to expand and modify the system to meet the changing demands. The information from the distribution study is used directly to determine how the demand for each demand region is distributed in proportion to the sales volume of each bulk plant.

The computerized part of the planning system is designed to be operated on an IBM 5100 desk top computer system equipped with a line printer and two tape cassette drives. The computer system is exclusively programmed in APL. In the subsequent sections of this paper, the computer-based system is presented:

1. In terms of the functions that its three major modules perform:
   - the Demand Projection System,
   - the Project Data System, and
   - the Cash Flow Projection System.

2. A general description of how the system is operated and its system elements.

3. A general description of its data files and programs (workspaces).

In order to enter and edit information on the data files in the system, only a rudimentary knowledge of APL is required. In order to operate the other system aspects (reporting, projection), no particular knowledge of APL is required. It is, therefore, expected that the planning analyst himself can utilize the computer-based portions of the planning support system thereby participating directly in the analysis.

Overview of System Operation

Demand Projection System

An overview of the process flow of the demand projection system is shown in Figure 2. At the beginning of the sequence of process steps are two programs (shown in the upper part of the figure) BASEDUTIL and PROJECTS, both within the workspace INIT on the program cassette DEMANDPROJ. These programs both allow for the definition, editing and reporting of data on the tape files.

The program PRO (workspace PROJECT) may contingently be executed. When it is run, it adds and permanently alters the content of the demand data on file BASED by adding in the demands of the energy projects on top of the absolute demands.

The last processing step in this system is program PDEMAND in workspace DEM. This program utilizes growth rates and base demands on tape file BASED to exponentially project the growing demands. These demands are then added to the absolute demands and stored on tape file REGDEM. This data file is used as source data for company cash flow projection.

When these processing steps are performed, they are automatically performed for all regions, all products and all years.

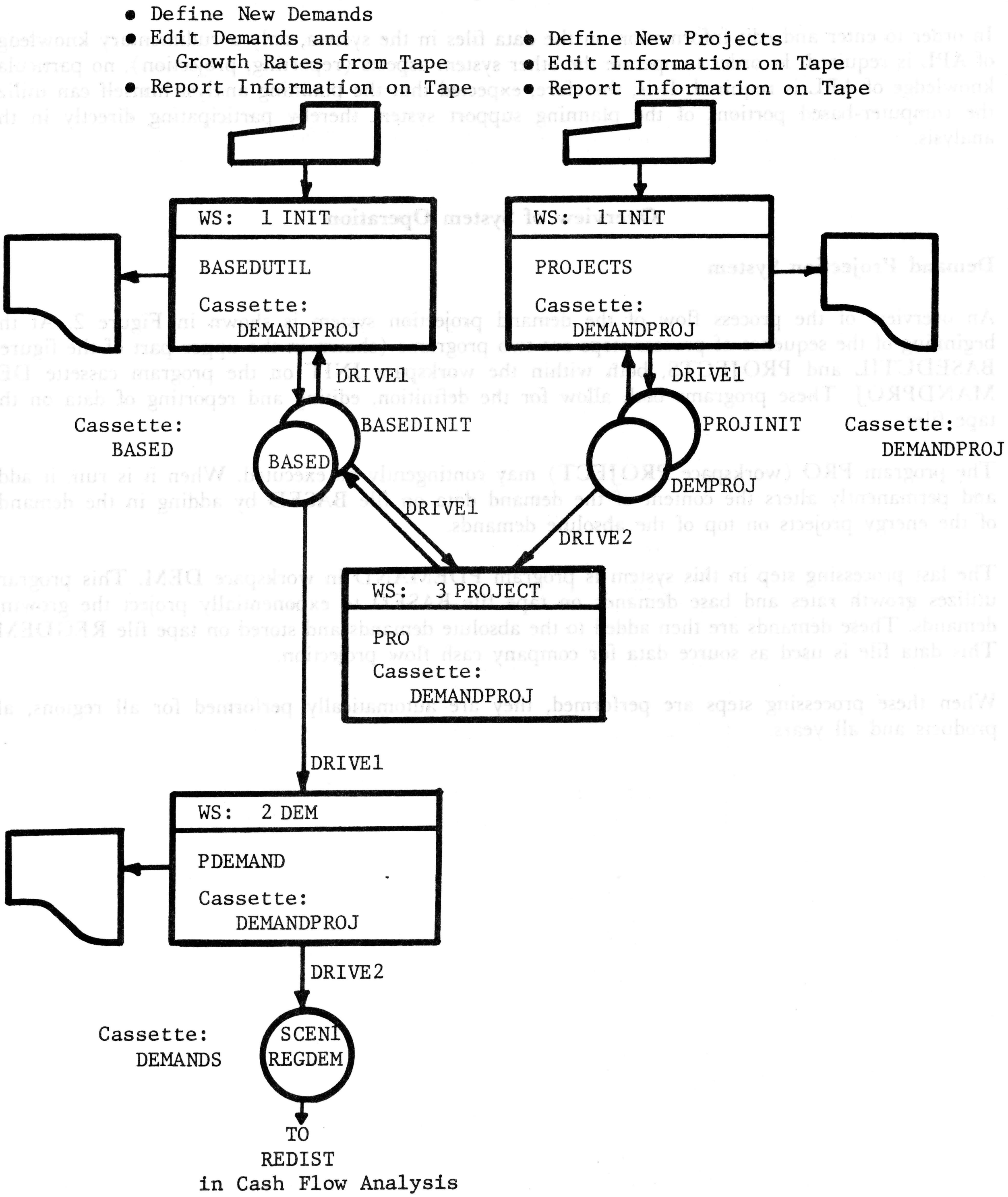

• Define New Demands
• Edit Demands and Growth Rates from Tape
• Report Information on Tape
• Define New Projects
• Edit Information on Tape
• Report Information on Tape
WS: 1 INIT
BASEDUTIL
Cassette: DEMANDPROJ
WS: 1 INIT
PROJECTS
Cassette: DEMANDPROJ
DRIVE1
DRIVE1
Cassette: BASED
BASEDINIT
BASED
PROJINIT
DEMPROJ
Cassette: DEMANDPROJ
DRIVE1
DRIVE2
WS: 3 PROJECT
PRO
Cassette: DEMANDPROJ
DRIVE1
WS: 2 DEM
PDEMAND
Cassette: DEMANDPROJ
DRIVE2
Cassette: DEMANDS
SCEN1
REGDEM
TO REDIST in Cash Flow Analysis

Figure 2 Overview of the Demand Projection System

Information System for Company Projects

The process flow for the basic company project information system is shown in Figure 3. These processing steps are quite simple although the program contains a number of options for project definition, editing and reporting.

The processing is (1) to provide project data from keyboard to the multiple cassette data files, and (2) provide reports (simple, selected or categorized) from the data files onto the printer.

The project data files are used within this system as well as used as source data for the company cash flow system.

- Define New Projects
- Edit Information on Tape
- Report Project Information on Tape

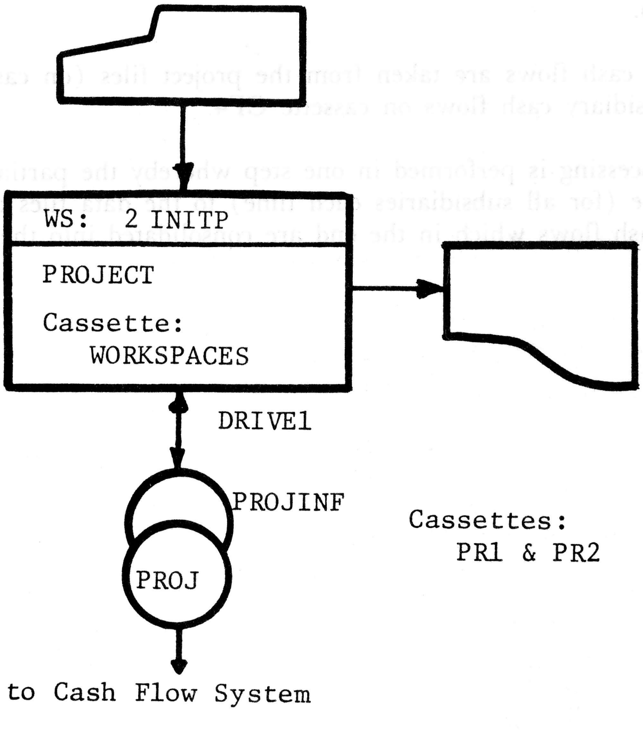

WS: 2 INITP
PROJECT
Cassette:
WORKSPACES
DRIVE1
PROJINF
PROJ
Cassettes:
PR1 & PR2
to Cash Flow System

Figure 3 Overview of the Company Project Information Definition

Company Cash Flow System

An overview of processing flow of the company's cash flow projection system is shown in Figure 4. This system consists of a set of four separate processing paths: (1) operating cash flow definition for each subsidiary, (2) revenue calculation, (3) transfer payment and markup calculations, and (4) project cash flow calculations. These four calculation paths result in separate data which are stored on separate cassettes (CF1, CF2, CF3, and CF4) as indicated. These data sets are then consolidated in a separate processing step into total subsidiary cash flows and into the consolidated company cash flow.

The operating cash flows are entered interactively from keyboard entries and may also be edited and reported separately. These functions are performed with the program LDOPCFL on each cassette CF1, 2, 3, 4 and 5. The basic operating cash flows are stored by subsidiary on tape CF1.

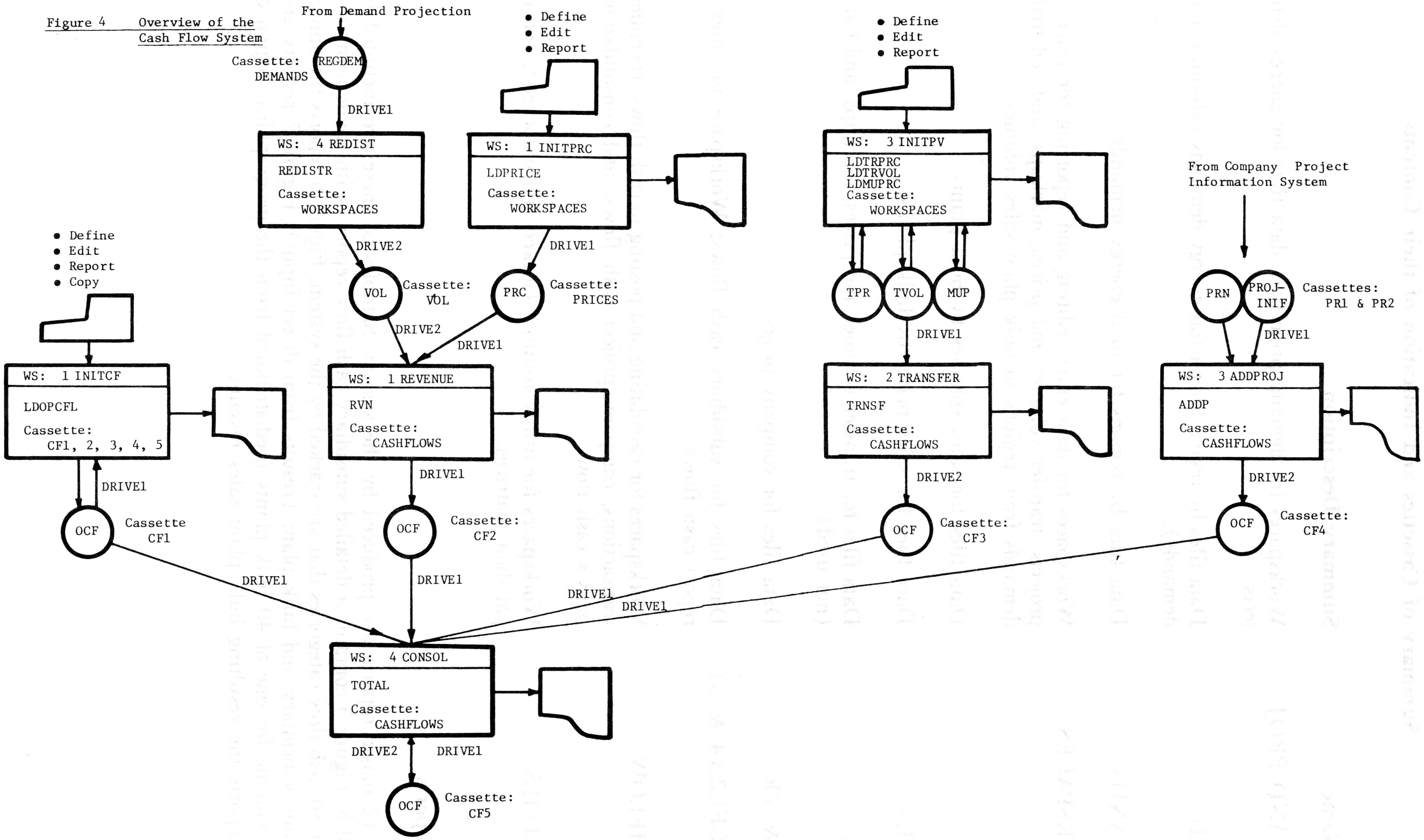
The revenues are calculated by (1) redistributing the demands by regions to sales volumes by bulk plants using program REDISTR with the sales volumes stored on file VOL, (2) inputting, editing and reporting product prices by bulk plants using program LDPRICE, with the prices stored on file PRICES, and (3) calculating the revenues from the data on these files, reporting on them, and storing the results on cassette CF2.

The transfer payments and markup calculations are performed through two processing steps: (1) inputting, editing and reporting data files for transfer volumes (TVOL), transferring prices (TPR), and markups (MUP), (2) calculating the transfer payments and markups by subsidiary and storing these on cassette CF3.

The company project cash flows are taken from the project files (on cassettes PR1 and PR2) and are posted to the subsidiary cash flows on cassette CF4.

The consolidation processing is performed in one step whereby the partial subsidiary cash flows are transferred one by one (for all subsidiaries each time) to the data files on cassette CF5 to produce the total subsidiary cash flows which in the end are consolidated into the company's own cash flow.

Figure 4 Overview of the Cash Flow System


From Demand Projection
Cassette: DEMANDS
REGDEM
DRIVE1
WS: 4 REDIST
REDISTR
Cassette: WORKSPACES
DRIVE2
VOL
Cassette: VOL
DRIVE2
• Define
• Edit
• Report
WS: 1 INITPRC
LDPRICE
Cassette: WORKSPACES
DRIVE1
PRC
Cassette: PRICES
DRIVE1
• Define
• Edit
• Report
• Copy
WS: 1 INITCF
LDOPCFL
Cassette: CF1, 2, 3, 4, 5
DRIVE1
OCF
Cassette CF1
DRIVE1
WS: 1 REVENUE
RVN
Cassette: CASHFLOWS
DRIVE1
OCF
Cassette: CF2
DRIVE1
• Define
• Edit
• Report
WS: 3 INITPV
LDTRPRC
LDTRVOL
LDMUPRC
Cassette: WORKSPACES
TPR
TVOL
MUP
DRIVE1
WS: 2 TRANSFER
TRNSF
Cassette: CASHFLOWS
DRIVE2
OCF
Cassette: CF3
DRIVE1
From Company Project Information System
PRN
PROJ-INIF
Cassettes: PR1 & PR2
DRIVE1
WS: 3 ADDPROJ
ADDP
Cassette: CASHFLOWS
DRIVE2
OCF
Cassette: CF4
DRIVE1
WS: 4 CONSOL
TOTAL
Cassette: CASHFLOWS
DRIVE2
DRIVE1
OCF
Cassette: CF5

Summary of Cassettes and Indication of their Contents

Cassette Name	Summary Description
1. DEMAND PROJ	Workspaces for demand projection, data files for energy demand projects.
2. BASED	Data files by region for partial energy demands (absolute and growing demands).
3. DEMANDS	Data files by regions for total energy demands.
4. WORKSPACES	Workspaces for data entry, editing and reporting for prices, company projects, transfer volumes and prices, markups, and redistributions of demands from regions to bulk plant sales volumes.
5. VOL	Data files for sales volumes by bulk plants.
6. PRICES	Data files for sales prices by bulk plants.
7. TRPV	Data files for transfer volumes and prices, markups, and barter factors (not used).
8. PR1 & PR2	Data files for company projects.
9. CF (CF1,2,3,4 & 5)	Data files for subsidiary cash flows; Workspace to enter, edit and report cash flows.
10. CASHFLOW	Workspaces for calculating and posting to cash flows: revenues, transfer transactions, company project cash flows, and consolidation of company's cash flow.
11. UTILITIES	Workspaces for utility routines (locked and unlocked), and for principal data matrices.

Sample Reports

A total of 12 major reports are produced by the system. Samples of these are included in Figures 5, 6, 7, and 8. Figure 5 contains a detailed subsidiary cash flow report (with random input numbers) for the over 50 cash flow categories that are tracked by the system. Figure 6 reports the annual product prices for one subsidiary and bulk plant (results from direct input). Figure 7 reports annual bulk plant sales volume for one of 40 bulk plants (redistributed from the geographical region demands). Figure 8 reports the resulting bulk plant sales revenue.

Figure 5

DATE OF REPORT- 14 3 1978 0

DETAILED CASHFLOW FOR SUBSIDIARY

IN MILLIONS OF .

		1978	1979	1980	1981	1982	1983	1985	1988
1	INFLOWS								
2	SALES REVENUE	45.000	750.000	840.000	1125.000	450.000	585.000	675.000	15.000
3	MISC. INCOME	32.000	76.000	99.000	43.000	10.000	1.000	46.000	35.000
4	SALES DISCOUNTS	68.000	2.000	56.000	32.000	41.000	79.000	54.000	87.000
5	NET SALES REVENUE	145.000	828.000	995.000	1200.000	501.000	665.000	775.000	137.000
6	P.I.F. LOANS	10.000	81.000	91.000	98.000	76.000	21.000	18.000	35.000
7	INVESTMENT REBATE	78.000	24.000	56.000	55.000	32.000	9.000	20.000	14.000
8	TOTAL INFLOWS	233.000	933.000	1142.000	1353.000	609.000	695.000	813.000	186.000
9	OUTFLOWS								
10	COST OF CRUDE FOR REFINING	47.000	76.000	51.000	1.000	52.000	61.000	29.000	94.000
11	COST OF PRODUCT PURCHASES								
12	CRUDE OIL (DIR.SALE)	51.000	41.000	8.000	27.000	37.000	25.000	88.000	10.000
13	RO	8.000	96.000	27.000	39.000	20.000	62.000	20.000	17.000
14	JO	37.000	1.000	77.000	35.000	10.000	47.000	97.000	18.000
15	LUBE	91.000	82.000	85.000	25.000	2.000	57.000	27.000	90.000
16	PETRO.	6.000	38.000	92.000	53.000	64.000	4.000	7.000	85.000
17	A	45.000	97.000	46.000	86.000	2.000	87.000	1.000	1.000
18	IMPORTS	63.000	66.000	75.000	64.000	26.000	60.000	5.000	50.000
19	SUM TOTAL CRUDE AND PRODUCTS	348.000	497.000	461.000	330.000	213.000	403.000	274.000	365.000
20	COST OF TRANSPORT								
21	MARINE TRANSPORT								
22	CRUDE FOR REFINING	69.000	42.000	29.000	87.000	25.000	71.000	43.000	49.000
23	CRUDE - DIRECT SALES	33.000	13.000	52.000	41.000	43.000	49.000	34.000	1.000
24	PROD.FROM	41.000	56.000	61.000	83.000	43.000	37.000	80.000	48.000
25	COASTAL MOVEMENTS	48.000	57.000	2.000	9.000	33.000	39.000	64.000	44.000
26	OTHER TRANSPORT								
27	PIPELINES	41.000	22.000	8.000	32.000	78.000	61.000	98.000	57.000
28	OWN TRUCKS	77.000	64.000	23.000	23.000	42.000	50.000	75.000	79.000
29	AGENT'S TRUCKS	12.000	78.000	83.000	5.000	96.000	82.000	57.000	54.000
30	RAIL	27.000	15.000	58.000	98.000	65.000	8.000	15.000	51.000
31	SUM TOTAL TRANSPORT COSTS	348.000	347.000	316.000	378.000	425.000	397.000	466.000	383.000
32	COST OF OPERATION								
33	MANUFACTURING	99.000	69.000	24.000	75.000	1.000	60.000	83.000	67.000
34	DESALTING	91.000	62.000	31.000	72.000	33.000	78.000	13.000	83.000
35	INSURANCE	86.000	28.000	79.000	88.000	69.000	79.000	24.000	78.000
36	FEES AND TAXES	93.000	99.000	98.000	92.000	82.000	70.000	42.000	63.000
37	DRY DOCKING	50.000	84.000	27.000	57.000	51.000	79.000	53.000	81.000
38	CONTINGENCIES	7.000	81.000	42.000	55.000	67.000	51.000	95.000	63.000
39	ADMINISTRATION	93.000	95.000	48.000	44.000	53.000	5.000	72.000	76.000
40	SPECIAL SURVEY	79.000	85.000	46.000	69.000	22.000	35.000	58.000	17.000
41	CONTRACTING LIABILITY	58.000	100.000	9.000	39.000	66.000	35.000	21.000	65.000
42	GENERAL OPERATIONS	44.000	5.000	29.000	23.000	37.000	31.000	19.000	84.000
43	MARKETING AND DISTRIBUTION	80.000	54.000	46.000	39.000	99.000	9.000	12.000	89.000
44	OTHER	56.000	76.000	90.000	29.000	91.000	17.000	75.000	84.000
45	SUM TOTAL OPERATION	836.000	838.000	569.000	682.000	671.000	549.000	567.000	850.000
46	SUM NET OPERATING CASH FLOW	¯1299.000	¯749.000	¯204.000	¯37.000	¯700.000	¯654.000	¯494.000	¯1412.000
47	CAPITAL EXPENDITURES	45.000	43.000	80.000	34.000	16.000	92.000	14.000	33.000
48	WORKING CAPITAL	96.000	55.000	39.000	14.000	93.000	38.000	16.000	87.000
49	INTEREST PAYMENTS	97.000	20.000	18.000	87.000	39.000	23.000	92.000	27.000
50	DIVID. TO	29.000	91.000	80.000	28.000	30.000	5.000	39.000	4.000
51	DIVID. TO OTHERS	23.000	54.000	34.000	60.000	69.000	47.000	56.000	61.000
52	DEBT REP. -	45.000	92.000	57.000	27.000	30.000	77.000	80.000	84.000
53	DEBT REP. - J	80.000	56.000	80.000	27.000	88.000	89.000	83.000	89.000
54	DEBT REP. - OTHERS	89.000	87.000	51.000	91.000	28.000	78.000	77.000	36.000
55	RENTAL PAYMENTS TO	26.000	70.000	77.000	100.000	84.000	61.000	14.000	40.000
56	TAX PAYMEMTS	22.000	34.000	99.000	99.000	14.000	45.000	57.000	91.000
57	SUM TOTAL OTHER OUTFLOWS	552.000	602.000	615.000	567.000	491.000	555.000	528.000	552.000
58	SUM NET CASH FLOW	¯1851.000	¯1351.000	¯819.000	¯604.000	¯1191.000	¯1209.000	¯1022.000	¯1964.000

Figure 6

```
REPORT OF ANNUAL PRODUCT PRICES BY SUBSIDIARY
----------------------------------------------

PRODUCT              PRICES    PRICES    PRICES    PRICES    PRICES    PRICES    PRICES    PRICES
 CLASS
                    PR. BBL.  PR. BBL.  PR. BBL.  PR. BBL.  PR. BBL.  PR. BBL.  PR. BBL.  PR. BBL.
                      1978      1979      1980      1981      1982      1983      1985      1988

 1 LPG               10.509    10.509    10.509    10.509    10.509    10.509    10.509    10.509
 2 PREMIUM GAS.      15.422    15.422    15.422    15.422    15.422    15.422    15.422    15.422
 3 REGULAR GAS.      12.401    12.401    12.401    12.401    12.401    12.401    12.401    12.401
 4 AVGAS            139.911   139.911   139.911   139.911   139.911   139.911   139.911   139.911
 5 JP4               26.758    26.758    26.758    26.758    26.758    26.758    26.758    26.758
 6 TURBO F. INTL     52.266    52.266    52.266    52.266    52.266    52.266    52.266    52.266
 7 TURBO F. DOM.     26.758    26.758    26.758    26.758    26.758    26.758    26.758    26.758
 8 KEROSINE          11.431    11.431    11.431    11.431    11.431    11.431    11.431    11.431
 9 DIESEL             4.929     4.929     4.929     4.929     4.929     4.929     4.929     4.929
10 MAR.DSL. INTL     40.335    40.335    40.335    40.335    40.335    40.335    40.335    40.335
11 MAR.DSL. DOM.     16.636    16.636    16.636    16.636    16.636    12.695    10.462    10.349
12 HEAVY FUELOIL      4.722     4.722     4.722     4.722     4.722     4.722     4.722     4.722
13 BUNKER C-INTL     33.302    33.302    33.302    33.302    33.302    33.302    33.302    33.302
14 BUNKER C-DOM.      7.992     7.992     7.992     7.992     7.992     7.992     7.992     7.992
15 CRUDE OIL           .000      .000      .000      .000      .000      .000      .000      .000
16 RC2 ASPHALT         .000      .000      .000      .000      .000      .000      .000      .000
17 ROOF ASPHALT      43.134    43.134    43.134    43.134    43.134    43.134    43.134    43.134
18 60/70 ASPHALT       .000      .000      .000      .000      .000      .000      .000      .000
19 MC1 ASPHALT         .000      .000      .000      .000      .000      .000      .000      .000
21 AUTOMOT. LUBE       .000      .000      .000      .000      .000      .000      .000      .000
22 AVIATION LUBE       .000      .000      .000      .000      .000      .000      .000      .000
23 MARINE LUBE         .000      .000      .000      .000      .000      .000      .000      .000
24 INDUSTRY LUBE       .000      .000      .000      .000      .000      .000      .000      .000
--------------------------------------------------------------------------------------------------
```

```
ANNUAL BULKPLANT SALESVOLUMES

PRODUCT               DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND    DEMAND
 CLASS              TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/  TH.BBLS/
                       YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR
                       1978      1979      1980      1981      1982      1983      1984      1985      1986      1987      1988

 1 LPG                 462.0     530.0     659.0     392.0     439.0     486.0     531.0     619.0     662.0     707.0     462.0
 2 PREMIUM GAS.       3238.0    3801.0    4498.0    4993.0    5739.0    6389.0    6853.9    7353.0    7669.4    7999.8    8345.0
 3 REGULAR GAS.        269.0     270.0     269.0     265.0     259.0     249.0     234.1     220.0     210.6     201.6     193.0
 4 AVGAS                 1.2       1.0        .8        .7        .5        .4        .3        .3        .3        .2        .2
 5 JP4                 139.0     173.0     208.0     260.0     230.0     232.0     234.0     236.0     242.5     249.2     256.0
 6 TURBO F. INTL       970.0    1215.0    1400.0    1610.0    1850.0    2125.0    2279.4    2445.0    2445.0    2445.0    2445.0
 7 TURBO F. DOM.       702.0     805.0     916.0    1042.0    1185.0    1355.0    1546.9    1766.0    2019.0    2308.3    2639.0
 8 KEROSINE            279.0     272.0     273.0     264.0     262.0     263.0     267.0     271.0     277.2     283.5     290.0
 9 DIESEL             7266.0    8889.0   10838.0   10550.0   11708.0   12320.0   13351.1   14095.0   14785.1   15508.6   16267.0
10 MAR.DSL. INTL      1615.0    2155.0    2580.0    1640.0    1725.0    1805.0    1897.6    1995.0    2096.4    2203.0    2315.0
11 MAR.DSL. DOM.       425.0     460.0     525.0     550.0     575.0     912.0     958.3    1407.0    1490.5    1579.3    1674.0
12 HEAVY FUELOIL       164.0     181.0     199.0     219.0     229.0     241.0     253.2     266.0     279.3     293.3     308.0
13 BUNKER C-INTL      6120.0    8215.0    9855.0    6380.0    6550.0    6860.0    7201.5    7560.0    7943.5    8346.5    8770.0
14 BUNKER C-DOM.      1615.0    1745.0    1975.0    2130.0    2180.0    2290.0    2407.0    2530.0    2656.9    2790.1    2930.0
15 CRUDE OIL              .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
16 RC2 ASPHALT            .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
17 ROOF ASPHALT         37.0      42.0      48.0      56.0      64.0      70.0      77.0      85.0      94.0     103.0     113.0
18 60/70 ASPHALT          .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
19 MC1 ASPHALT            .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
20 TOTAL ASPHALT          .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
21 AUTOMOT. LUBE          .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
22 AVIATION LUBE          .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
23 MARINE LUBE            .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
24 INDUSTRY LUBE          .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
25 TOTAL LUBE             .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0
26 NATURAL GAS        5160.0    7030.0   12950.0   13930.0   18030.0   22070.0   23839.1   25750.0   25750.0   25750.0   25750.0
27 ETHANE                 .0        .0        .0        .0        .0        .0        .0        .0        .0        .0        .0

TOT DISTILLATES      14904.2   18041.0   21507.8   21174.7   23533.5   25650.4   27622.6   29788.3   31236.0   32778.6   34424.2
TOT OIL PRODUCTS     23265.2   28712.0   34195.8   30295.7   32931.5   35527.4   38015.3   40763.3   42777.7   44915.5   46894.2
TOT HYDROCARBONS     28425.2   35742.0   47145.8   44225.7   50961.5   57597.4   61854.4   66513.3   68527.7   70665.5   72644.2
------------------------------------------------------------------------------------------------------------------------------------
```

```
                     SALES REVENUES BY PRODUCT

PRODUCT             REVENUES  REVENUES  REVENUES  REVENUES  REVENUES  REVENUES  REVENUES  REVENUES  REVFNUES  REVENUES  REVENUES
 CLASS
                       YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR      YEAR
                       1978      1979      1980      1981      1982      1983      1984      1985      1986      1987      1988

 1 LPG                 4.855     5.570     6.926     4.120     4.614     5.107     5.580     6.505     6.957     7.430     4.855
 2 PREMIUM GAS.       49.936    58.619    69.368    77.002    88.507    98.531   105.701   113.398   118.277   123.373   128.697
 3 REGULAR GAS.        3.336     3.348     3.336     3.286     3.212     3.088     2.903     2.728     2.612     2.500     2.393
 4 AVGAS                .168      .140      .112      .098      .070      .056      .048      .042      .037      .032      .028
 5 JP4                 3.719     4.629     5.566     6.957     6.154     6.208     6.261     6.315     6.488     6.667     6.850
 6 TURBO F. INTL      50.698    63.503    73.172    84.148    96.692   111.065   119.135   127.790   127.790   127.790   127.790
 7 TURBO F. DOM.      18.784    21.540    24.510    27.882    31.708    36.257    41.392    47.255    54.025    61.765    70.614
 8 KEROSINE            3.189     3.109     3.121     3.018     2.995     3.006     3.052     3.098     3.169     3.241     3.315
 9 DIESEL             35.812    43.811    53.417    51.998    57.705    60.722    65.803    69.470    72.871    76.437    80.175
10 MAR.DSL. INTL      65.141    86.922   104.064    66.149    69.578    72.805    76.541    80.468    84.559    88.858    93.376
11 MAR.DSL. DOM.       7.070     7.652     8.734     9.150     9.566    11.578    12.139    14.720    15.538    16.312    17.324
12 HEAVY FUELOIL        .774      .855      .940     1.034     1.081     1.138     1.196     1.256     1.319     1.385     1.454
13 BUNKER C-INTL     203.809   273.578   328.193   212.468   218.129   228.453   239.826   251.765   264.538   277.958   292.060
14 BUNKER C-DOM.      12.908    13.947    15.785    17.024    17.424    18.303    19.238    20.221    21.235    22.300    23.418
15 CRUDE OIL            .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
16 RC2 ASPHALT          .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
17 ROOF ASPHALT        1.596     1.812     2.070     2.416     2.761     3.019     3.321     3.666     4.055     4.443     4.874
18 60/70 ASPHALT        .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
19 MC1 ASPHALT          .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
20 TOTAL ASPHALT        .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
21 AUTOMOT. LUBE        .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
22 AVIATION LUBE        .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
23 MARINE LUBE          .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
24 INDUSTRY LUBE        .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
25 TOTAL LUBE           .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
26 NATURAL GAS          .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000
27 ETHANE               .000      .000      .000      .000      .000      .000      .000      .000      .000      .000      .000

TOT HYDROCARBONS      460.20    587.22    697.24    564.33    607.43    656.32    698.81    745.03    779.42    816.05    852.35
```

Discussion

The use of the APL environment for this application proved very successful. The applications can be estimated to have an APL suitability score of 28 (Schengili, 1978). The major incentives for using APL for this application are:

- Mandatory need for user interaction,
- Medium transaction volume and CPU load,
- Intermittent run frequency,
- Good availability of APL implementations,
- Urgent system completion, and
- Complex logic and systems data flow.

As can be expected, APL provided the required flexibility in systems implementation since parts of the system had to be reprogrammed twice or three times before the final analyses could be provided.

The major problems associated with implementing an analysis support system in parallel with a major study of this kind are that (1) the total study timeframe is very short, (2) the organizational and accounting structure of the firm is not known in advance, and (3) most importantly, the total availability and resolution of data are not known until the data actually have been collected and analyzed. It would be our conclusion that this kind of analysis system could not be implemented in any other environment (e.g., FORTRAN, PL/I) without increasing the implementation effort by perhaps a factor of 5.

The use of a portable minicomputer provided a good environment for their applications and provided the ability to bring computer facilities on-site. Its configuration consisted of 64K "core", 2 tape drives and a 120 cps wide carriage printer. The oil company does not have other computer facilities available on this site and, therefore, it was required to supply separate resources. The IBM 5100 computer was purchased and carried on-site as air luggage and was started up and running within hours after arrival.

This computer environment is quite slow (separate tape files for each geographic area, for example) and requires about 2 hours to calculate and print the reports for the revenue calculating portion of the cash flow analysis. That, however, does not provide any obstacle to its usefulness since a total cash flow analysis can be completed in less than 2 days as compared to weeks using manual methods. There is no doubt that future versions of the portable computers with floppy discs (e.g., IBM 5110) and CPU's with 3-5 times the internal speed will enhance the environment extensively and make it quite competitive with much larger systems.

Reference

1. Schengili, Josef, "Checklist Shows if APL is Suitable Language", Computerworld, May 8, 1978, p.34.

EXPERIENCE WITH DIRECT DEFINITION ONE-LINERS IN WRITING APL APPLICATIONS

Donald B. McIntyre
Department of Geology
Pomona College
Claremont, California

Abstract

Iverson, Orth, and others have advocated a condensed form for writing function definitions that they call **direct definition**. This not only automates some aspects of the task of defining functions, but also forces each definition to be complete within a single line. The use of direct definitions often forces a restructuring of the content in ways that may at first seem unfamiliar, but which, when pursued and developed, greatly simplify both the functions and the accompanying documentation. The paper is illustrated with examples taken from applications in geology.

Introduction

Unlike other programming languages whose purpose is to help the programmer gain control of a machine, APL was intended from the start to be an aid to human thought. It is APL's stimulus as an intellectual tool that accounts for its extraordinary appeal to a wide spectrum of users. Those who have problems to solve in their own fields, but perhaps do not have time or inclination for the study of computing machinery, are delighted to find that, like mathematics, APL is a language well adapted for succinct and formal description of complex systems, which, with its aid, can be taken up and held together in the mind (that is, comprehended), possibly for the first time.

As in mathematics, an expression once formulated in APL can often be reduced and simplified, with clarification of the thought it embodies. But APL does more; for it revises and extends mathematics. Its consistent syntax and its recognition of inverse functions (such as ⊥ and ⊤) and of dualities (such as ∧ and ∨ with respect to ~) provoke thought and may suggest solutions. In place of separate boolean, linear, and other algebras, APL provides a single language of great power and flexibility. Its creation was an important step in the continued evolution of mathematical notation.

So accustomed are we to the standard symbols of mathematics that we learn with surprise about the slow and laborious progress of their introduction and acceptance. For example, William Oughtred, who had introduced the St. Andrews cross (×) for multiplication in 1631, wrote as follows 16 years later: "...Which Treatise being written...with symboles or notes of things instead of words, seemed unto many very hard; though indeed it was but their owne diffidence, being scared by the newness of the delivery; and not any difficulty in the thing it selfe. For this symbolicall manner ...plainly presenteth to the eye the whole course and processe of every operation and argumentation."

Early computers being small and slow, the programmer took seriously his responsibility to write programs that made best use of available space and time, and considerable human effort and ingenuity were often expended so that a single instruction could be eliminated. The machine was paramount, and the more the programmer knew about it and was concerned with it, the better he could do the job expected of him. Moreover, turnaround time was so long that programs had to print every result that might conceivably prove useful at a later time. It is hardly surprising that in this environment programs were commonly large packages in which the fundamental algorithms were concealed under a superstructure with loops and branches and declarations, which were of little interest to the user, however necessary they might have been for running the machine.

APL, and the Iverson notation out of which it grew, can and should be used at the desk, on the blackboard, and in books, quite independently of any available implementation on a machine (see, for example, McIntyre, 1969). But, of course, without a computer the large program, and all it stands for, is useless. On the board we write and evaluate expressions and we define and use simple functions, and the thesis of this paper is that when we do use a computer, we **should not write programs** (in the traditional sense of a complex, many-lined algorithm with loops, branches, and subroutines), but instead, **should continue to define simple functions**, each one corresponding to an essential and distinct concept in our subject matter.

Just as we learn in good sentence structure not to put too much or too little into a sentence or a paragraph, so in APL serious thought should be given to the appropriate selection of functions. The case for this style in APL has been eloquently stated by Paul Berry and his colleagues (Berry, & others, 1971) in an article that should be read and reread by everyone involved in computing.

> "We are advocating a style of programming in which the programmer's first responsibility is to find what are the fundamental ideas, and then to provide a set of functions which have a direct correspondence to those ideas. ... A topic is best treated by a family of simple functions, rather than by a single monolithic program. ... The key to the use of APL in education is the consistency with which functions are treated, and the possiblity of choosing functions so that they not only get calculations performed, but also correspond psychologically to the concepts out of which the student builds his understanding."

Characteristics of Good APL

Whereas many languages impose structure on the program, in APL discipline comes only from the writer, who has full responsibility for choosing the functions and determining how they interact. It is easy to abuse a powerful tool, and APL, which offers the writer such complete freedom, is no exception. When APL is badly written, perhaps no other language is so difficult to read. This is why some attempts have been made to enforce structured programming in APL. Indeed I have heard professional programmers, driven to despair on inheriting poor APL, claim that its documentation is never adequate and that no one else's APL can be deciphered without more work than is required to complete the application one's self in the first place.

These are serious criticisms and it is important to recognize that too often they are justified. But the cure is not to write long programs within which elaborate embedded comments distract the reader and interfere with the proper continuity of the algorithm. And neither is it a cure, as Berry and others have pointed out, to write programs with many very short lines; for, like a paragraph of extremely short sentences, it is then difficult to grasp at one time a sufficient number of relationships to satisfy the mind.

Although good style, whether in English or in APL, may defy useful definition, and may be impossible to canonize in a set of rules for a student to follow, it is nevertheless true, though paradoxical, that such style is recognized with assurance and pleasure by an eye educated through careful and reflective study of fluent writing. We must learn to read critically both our own work and that of others, developing the habit of rephrasing expressions in order to choose the clearest and most pleasing form. When a student says he feels sure a particular APL function could be better written, he is demonstrating that he already knows something about style.

While recognizing that this cannot be reduced to rules, we can nevertheless isolate certain aspects that, when kept in mind, contribute to the writing of what I believe to be better APL:

1. Selection of Functions

If we are to treat the topic, as we should, by a set of simple functions, the initial problem is to determine what these functions are to be. It is vital that the selection be made with care and with adequate knowledge of the structure of the subject, because it is at this stage that the whole system is defined. Commenting that "Sometimes the quest for an appropriate set of building blocks is a subtle and difficult task", Berry and others (1971) suggest that the verbs occurring in a brief English description of what has to be done will be suitable candidates for defined functions. I find this advice sound and helpful. I must admit, however, that I sometimes find it easiest in addressing a problem to write initially a program that works, with as many lines as is convenient, and subsequently to break this down into logical parts. But this may well be no more than admitting that I first write a program and only think about it afterwards. If so, it is difficult to commend the practice. But perhaps the creation of a working program is legitimate for someone who does not have a theoretical and abstract turn of mind. What is important is not to stop there with such an aid, but to use the original program as a rough draft to be worked upon.

2. Function Names

If a function plays the same role in APL as a verb does in English, then evidently the verb is the function's best name. Short names or mnemonic abbreviations are, however, a great help; for otherwise, expressions in which defined functions appear become clumsy, losing the advantage of their symbolism. Moreover, a long name imposes a burden when space or processing time are limited and, as in the creation of an incidence matrix, we look for the occurrence of function names in strings representing other functions. It is a help, too, if function names look different from variable names, and because the latter are generally single letters or words describing and clearly identifying the variable, I prefer in most cases to use abbreviations of two or three letters.

At one time I used simple function names like *RAD* and *DEG*, which seem reasonably obvious abbreviations of radians and degrees, especially in the content of trigonometric expressions. But such names are ambiguous: is it *RAD*'s argument or its result that is in radians? Iverson, on the other hand, has used the form *DTR* for **D**egrees **T**o **R**adians and *RTD* for **R**adians **T**o **D**egrees. These names are not only unambiguous as indicators of the direction of their function, but the inverse relationships of a pair of functions is at once evident. Paul Berry has pointed out to me the advantage of the form *RFD* (for **R**adians **F**rom **D**egrees) whose sequence of letters tells us that the argument (on the right) is in degrees and the result (on the left) in radians. I have used this form to advantage in sequences such as *CFP PFL*, for **C**artesian-**F**rom-**P**olar **P**olar-**F**rom-**L**atitude/**L**ongitude, which combine much better than *PTC LTP* do.

I have distinguished certain classes of functions by a special prefix on the function name. For example, functions that are part of the compiler used with directly defined functions are identified by the prefix ⍙. The user will normally want to ignore these functions, treating them as if they were part of the machine. On the other hand, functions like an arithmetic progression vector, common to many applications, may be thought of as extensions to the set of APL's primary functions and by identifying them with the prefix Δ, the user can arrange to suppress them, as he does + and ×, from the list of functions used in a particular application.

Each workspace or group of functions should be provided with a function that lists the function names, giving for each a one-line statement that expands and explains the mnemonic significance of the name. I like to use a function that, for each function, provides a one-line statement, and an illuminating example with typical data; for example:

```
    ∆APV ARITHMETIC PROGRESSION VECTOR

    ∆APV:ω[1]⌊ω[0]+α×⍳⌈1+('-'PWω)÷α

     .3 ∆APV 2 6
2  2.3  2.6  2.9  3.2  3.5  3.8  4.1  4.4  4.7  5  5.3  5.6  5.9  6
```

On an IBM 5100, the illustration can be executed after it is scrolled to the active line. I refer to this procedure as "interactive documentation", and make constant use of it in my work and teaching. The ease with which information on the screen can be brought to the active line for execution is an important advantage of the IBM 5100, and I miss it when I have to use another system. For more complex applications I provide a multi-lined function that serves, with data included in the workspace, as an example of the way in which the various functions can be used separately and together.

3. Side Effects

Clarity is possible and confusion can be avoided only if each function performs as it pretends to do. The first condition is therefore that the arguments and the results are explicit. We are misled by electrical apparatus whose leads are apparently displayed, but which has a secret wire reaching it through a hole in the table on which it stands. Similarly, if the result of a function depends not only on the arguments stated but on some other variable, the function will fail to convey a clear statement of its purpose and action. Berry and others (1971), cogently protesting against the use of global arguments and global results, which they appropriately call "side effects", point out that our "ability to use defined functions in new expressions depends upon the fact that there are explicit results which may be passed directly to the next function as its argument."

At the Los Angeles Conference on the History of Computing Languages, held in June 1978, F.P. Brooks suggested that in retrospect it was perhaps a mistake that local rather than global variables are specified in the header of an APL function. But one can go further than this, and ask whether an APL function should ever be permitted to contain global variables. Although strongly inclined to the view that they should not, I admit that, after encountering cases where exclusion of global variables seemed sufficiently difficult, I arranged to make variables global whenever their names contained an underscored character.

This seeming necessity for global variables arose in two contexts: (a) when an environment is being created for future work, as in the definition of scaling parameters that will be applied to various sets of data before plotting, or in analogous situations where the user simply does not need or wish to know about coefficients that are computed and used in some transformation; and (b) when it appears that more than two arguments are required by a function. I am now of the opinion that both situations can and should be avoided. With expenditure of more thought, new arrangements can be devised that remove the inconvenience of the first case, and that eliminate the apparent necessity of the second.

4. The Line as a Comprehensible Thought

There is no rule for determining the best length of a sentence, and none that gives the proper length of an APL expression. A series of short lines, each with little content, is tiring and difficult to read. On the other hand, it is a mistake to suppose that a short APL expression is necessarily easy to understand; for the thought it embodies, like Einstein's concise statement of the relationship between energy and mass, may be profound, and there are, of course, added difficulties when the reader is not sufficiently familiar with APL's primary functions or common constructs. The important thing is to review each line critically and ask whether it has too much content to be comprehended as a single thought, or whether, on the other hand, it would make a clearer and more complete expression if it incorporated one or more adjoining lines. Long lines tend to alarm, and one should not arbitrarily string unrelated expressions into a single line by using constructs like, `... ,0ρ ...`. But long lines can be legitimate, and if well written they need not be particularly difficult to read. They should be read both from the left to right and from right to left to ensure that they are comprehensible.

It is often surprising how much an expression can be simplified. Redundant parentheses should be removed, unless included to bring out the symmetry of an expression, or for some other good reason. Instead of subtracting a positive number at the right of the expression, we can add a negative number at the left, thereby eliminating the need for a pair of parentheses. It is difficult sometimes to escape from the original way we approached a problem; so we find expressions containing phrases such as `(÷X)× ...` or `... ×÷X` that have never been properly assimilated. Again, with the aid of Iverson's extensions of De Morgan's law we can simplify logical expressions, such as:

```
      (~U∧1⌽U←S=' ')/S←' NOW   IS   THE TIME   FOR '
NOW IS THE TIME FOR

      (U∨1⌽U←S≠' ')/S
NOW IS THE TIME FOR
```

Although the choice between alternative forms of expression may be a matter of taste, the writer should at least give thought to the consistency of his usage and to the advantages and disadvantages of each possibility before making a final decision. Consider for example `0⊥ρV` for `¯1↑ρV`, or `V[⍳3]× ...` for `(3↑V)× ...`;

or the choice between the following:

```
      I∘.=I←⍳3
      (⍳3)∘.=⍳3
      1 2 3 ∘.=1 2 3
      'ABC'∘.='ABC'
```

A function with a long list of local variables should be considered as a candidate for subdivision. It probably incorporates several distinct concepts, and if so, this will become obvious if its action is described in words and attention is given to the verbs. If variables within a function are given specified values, this should be done at the extreme left of the line. Specifications embedded within a line can be a source of frustration because they are easily overlooked by a reader trying to discover how a variable acquired its value. This is especially true when a variable name is used with a new and unrelated meaning. However, when a variable belongs to one line only and is not used subsequently, its specification should be included within that line.

5. Generalization to Arrays

One of the principal sources of APL's power and beauty is the ease with which arrays can be treated as entities. In this respect, APL extends and implements the Algebra of Multiple Quantity about which J.J. Sylvester, who coined the term "matrix", wrote so eloquently a century ago. Consequently, we should always examine our functions to see whether they are properly generalized. They should not be written for scalar arguments when they could be written for vectors, nor for vectors if they could easily be extended to matrices. There are, of course, exceptions; for example, the extension beyond three-dimensions in functions written for crystallography or structural geology would probably be a distraction serving no useful purpose.

The loops characteristic of Fortran can often be replaced by a vector in APL, and a nested loop by a matrix. While loops cannot always be avoided, every one needs examination and justification. But the principal advantage of generalizing our functions is intellectual; the exercise helps us think at the higher level that Sylvester advocated.

6. Phrases and Idioms

In every language there are phrases that express commonly occurring concepts with particular elegance. These are the idioms that add so much power when used by someone conversant with the language, but which are responsible for many of the difficulties met by beginners who do not recognize them. The fluent reader, instead of taking single words in isolation, sees the idioms as entities. In reading a function this is analogous to the generalization from scalars to arrays in the analysis of data.

Perlis and Rugaber (1977) have published a useful list of APL idioms. Anyone who studies this will increase his fluency both in reading and writing. Here are a few examples of phrases that should be recognized as idiomatic units when found embedded within an expression:

a) `V∘.+Nρ0` or `V∘.×Nρ1`

Because 0 and 1 are the identity elements for addition and multiplication, these expressions cannot change the values given by `V`. The result is a reshape of `V`.

b) `1⋆M[;0]`

Because 1 to any power is still 1, the right argument provides the shape of the resulting vector of 1's.

c) `U∨1⌽U`

The domain of `∨` requires `U` be a logical vector; that is, a vector containing 1's or 0's. The result is a selection vector differing from `U` in that for every subset selected by `U`, the new vector will select an additional (cyclically) preceding element.

d) `U\U/M`

If `M` is a matrix, the result is that the columns of `M` specified by the 0's in `U` will be replaced by 0's.

e) `~X∊Y`

The result, with the same shape as `X`, will have 1's indicating elements of `X` that are not members of `Y`, irrespective of the shape of `Y`.

f) `+/∧\M=' '`

`∧\` will set to 0 all elements encountered after the first 0. The result is therefore the number of leading spaces in each row of the matrix *M*.

g) `(1-(A=' ')⊥1)⌽A`

If *M* is a logical matrix, `M⊥1` extends the right argument to `(0⊥ρM)ρ1` or `1⋆M[0;]` and gives the result of 1 plus the number of elements following the last 0 in each row of *M*. Consequently, the above expression will right-justify the character matrix *A*.

Idioms, as Perlis and Rugaber point out, are candidates for new primary functions. Indeed, I have thought it convenient to define a special set of functions for some of them. But it can be argued that replacing idioms within a function by new names, whose meaning must be remembered or looked up, is an annoyance to those who already know the idiom, and of little help to those who do not. In deciding whether to dignify a phrase by defining it as a function, we should consider (a) whether the phrase is used in many functions or in only one; (b) whether the function containing the phrase will be simplified or perhaps made harder to read if the phrase is separated; and (c) whether the phrase is sufficiently interesting by itself in the context of the subject matter.

In a set of functions involving trigonometric relations, should the expression `○X÷180`, which computes radians from degrees, be embedded as a phrase or defined as a separate function? It is not easy to decide the question; for, while the phrase is certainly useful in connection with many other functions, its name as a function will not be much shorter than the expression itself. There seems no escape from the conclusion that it requires fine judgement based on thoughtful experience to know what one should do in such instances.

Direct Definition of Functions

APL provides a comprehensive set of primary functions and permits the definition of secondary functions through what is called ∇ mode. Once defined, these secondary functions can be used like primary functions in most APL expressions. They begin with a header establishing the syntax, declaring local variables, and identifying the names of the function's explicit arguments and result, if any. There is no limit to the number of lines that may then follow.

In his "Elementary Analysis", K.E. Iverson (1976) introduced a new form of function definition, called direct definition, used also by D.L. Orth (1976) in his "Calculus in a New Key". This form is very simple: the name of the function is separated by a colon from an APL expression in which α symbolizes the first and ω the second argument. The explicit result is always the result of this expression; consequently there is no need for the customary header. In the following example, the colon can be read "is"

```
      RFD:○ω÷180

      RFD 57+¯1×⍳10
0.99484  0.97738  0.95993  0.94248  0.92502  0.90757  0.89012  0.87266
      0.85521  0.83776
```

Iverson also introduced a second form of direct definition in which three APL expressions are separated by colons: a primary expression; a proposition with the result 0 or 1; and a secondary expression. The proposition is evaluated first, this being followed by either the primary or the secondary expression, depending upon whether the value of the proposition is `0` or `1`. This form provides a conditional statement which permits easy definition of recursive functions. Indeed the reaction of students and colleagues is always that never before have they been able "to think recursively". As an example, consider the problem of finding names in a string. If the names can contain numeric characters but must begin with an alphabetic character, and if *A* is the alphabet `'ABC ... Z'`, *W* is the vector of digits `'012 ... 9'`, and *S* is the string, then `S∊A` and `S∊W` are vectors pointing to alphabetic and numeric characters, respectively. Then the following recursive function will provide a **M**ask for the **N**ames, excluding digits that are not part of legitimate names:

```
      MNM:X MNMω:α∧.=X←α∨ω∧¯1⌽α:α

      ((S∊A) MNM S∊W)/S←'6 NAME99 99'
NAME99
```

The system of direct definition is especially suitable for work at the desk or on the board. To use it on a computer, it is, of course, necessary to have a compiler that translates from the form of direct definition to executable APL. Iverson (1976, 1978) and Orth (1976) have made such a compiler available.

Illustrations

The following examples of directly defined functions are chosen to illustrate points made in the preceding text. They use zero origin.

1. The Mann-Whitney test

This non-parametric rank-sum test is used to determine whether the difference between the means of two samples is significant. After combining the data into a single sample, we rank the values in increasing order of magnitude. If more than one value has the same rank, we assign each of the tied values the mean of the ranks that they jointly occupy. The rank-sum, *R*, is then computed as the sum of the ranks assigned to the values of the first sample. The subsequent steps, being less interesting, are not included in the illustration.

The following is a portion of an APL function in which the samples are *A* and *W* and the index origin is 1.

```
     ∇ Z←A RF W
[1]    Z←A,W
[2]    I←⍋Z
[3]    Z←Z[I]
[4]    M←((Z≠1⌽Z)/Z)∘.=Z
[5]    R←+/[1] M×⍉(⌽⍴M)⍴(+/M×(⍴M)⍴⍳⍴Z)÷+/M
[6]    R←+/(I≤⍴A)/R
     ∇
```

In conformity with the rule that a variable will be specified at the left of a line if it is used in a subsequent line, the first three lines have not been combined into a single line. The fourth line can be simplified by interchanging the arguments of `∘.=`, though this will cause *M* to be transposed. But the fifth line needs attention. Twice in that line a sum reduction, `+/`, is immediately to the left of the combination of two matrices by the function times, `×`. This sequence should be replaced by matrix multiplication, `+.×`. The six lines can then be written as a single directly defined function, whose first argument is `⍴A` and whose second is `A,W`:

```
RF:+/(α≥I+1)/M+.×((1+⍳⍴ω)+.×M)÷+⌿M←ω∘.=(ω≠1⌽ω)/ω←ω[I←⍋ω]
```

The phrase `(ω≠1⌽ω)/ω←ω[I←⍋ω]` ranks the values and selects the unique grade numbers. *M* is then the logical matrix that shows the distribution of grade numbers amongst the successive values. If there are no ties, then *M* is an identity matrix; otherwise the number of ties is the number of 1's in any column of *M*, that is `+⌿M` is the vector of divisors for the means of the ranks.

`(1+⍳⍴ω)+.×M` is the vector of dividends for the means of the ranks; that is, each element is the sum of the tied values for a given rank.

The leftmost `M+.×` distributes the means into a vector whose length is `⍴ω`. The rank-sum, *R*, is then found as the sum of the ranks corresponding to the first `α` ungraded values; that is, the sum of the ranks for the first sample.

The resulting line is not too long, because its parts fit together in a meaningful way, and the specification of local variables, which are used only in this line, also makes sense. The function is not only easy to read, but it corresponds directly to the core of the Mann-Whitney test, namely to the computation of the rank-sum for the first sample. Loops are avoided by use of the logical matrix, *M*, whose key role in the algorithm stands out in the definition. Moreover, because the computer stores this logical matrix in bits rather than in double words, much larger samples can be tested than was possible by the original six-line function. As often happens, in striving for clarity, we have also obtained a large gain in efficiency.

2. Incidence Matrix of Functions Used to Define Functions

A function may refer to other defined functions in its definition, and it is said to be recursive if it appears in its own definition. When analyzing a system of functions, it is useful to prepare an incidence matrix, with each row indicating the functions used by a given function, and with each column indicating the functions which make use of a given function.

Let the names of defined functions be rows of matrix *A* and the corresponding function definitions be rows of matrix *W*. Then the heart of the solution lies in the logical matrix `A∘;=W`. Now because `ρA∘.=W↔(ρA),ρW`, the size of the array can be large. To cut down time and space requirements for an IBM 5100, we first edit *W*, so that all characters except possible function names are replaced by spaces. Then we compress by removing redundant spaces. The functions used in editing are interesting, but in this illustration we suppose the editing already done.

The function *REF*, for cross references, includes the two following lines:

```
U←W∨.≠' '

Z←(' ',A,' ')FIND U/[0]' ',W,' '
```

The compression by *U* eliminates rows that are all blank. We ensure that there is at least one space before and after each name, and then we use the function *FIND* to identify the function names that appear in the rows of *W*. As originally written, *FIND* was as follows:

```
    ∇ Z←A FIND W;B
[1]    B←U\[0]⌈/+/[2]∧\[2]((ρB)[0 1 2]ρι0⊥ρA)⌽[3] B←1 2 0 3 ⍉A∘.=W
[2]    Z←B≥(0 0 +1↑ρA)ρ2++/∧\ 0 1 ↓A≠' '
    ∇
```

If `ρA` is `3 4` and `ρW` is `2 8` then `ρA∘.=W` is `3 4 2 8` and `ρ 1 2 0 3 ⍉ A∘.=W` is `2 3 4 8`.

The dyadic transpose is a powerful feature of APL, and one that is worth the trouble to master; but in this example it was used so that the last two axes relate to the longest name and the longest definition. That made it easy to specify the rotation that makes it possible for us to identify function names appearing in function definitions. But this was not necessary. We can define *FNS* (**F**unction **N**ames) to do the same job as line 1 of *FIND*, but without the dyadic transposition:

```
FNS:⌈/+/[1]∧\[1]((3↑(ρα),ρω)ρ(ι0⊥ρα)∘.+(1↑ρω)ρ0)⌽α∘.=ω
```

Although the expression for the rotation is now more complicated, I believe this is worth the gain of eliminating the re-arrangement of the matrix `α∘.=ω`. If the line looks forbidding, we should, as a first approximation, read it as:

```
⌈/+/∧\X⌽A∘.=W
```

The rotation (here indicated by *X*) is chosen to produce an alignment of 1's if the name (a row of *A*) is present within a row of *W*. The phrase `+/∧\` gives the number of 1's before the first 0, thus discounting spurious or partial matches. `⌈/+/∧\` is therefore the longest substring matched. The question whether it is long enough to match the name in *A* is answered in another function, *ICM* (**I**ncidence **M**atrix):

```
ICM:((2ρ1↑ρα)ρ2++/α≠' ')≤U\(' ',α,' ')FNS(U←ω∨.≠' ')⌿' ',ω,' '
```

At first it had seemed impossible to avoid *U* being global to function *FIND*, because *FIND* already had two arguments. But *U* is now localized in a single, directly defined function. This function, *ICM*, prepares the arguments for *FNS*, expands the result of *FNS* and applies the appropriate test. *FNS*, on the other hand, contains the essential algorithm without extraneous housekeeping details.

We now have two directly defined functions instead of one function of two lines. As a result, *U* now disappears from the main function, *REF*, and instead we have the simple line:

```
Z←⍉A ICM W
```

Z is the first-level incidence matrix; that is, it identifies the functions directly referred to by other functions. The second-level incidence matrix identifies in addition those functions which, while not themselves referred to directly, are employed by functions that are referred to directly. This second-level incidence matrix is defined as:

```
Z←Z∨Z∨.∧Z
```

In practice we continue on to third and, if need be, higher levels, repeating this procedure until *Z* is unchanged.

3. Integration by Simpson's Method

Integration by Simpson's method is possible when the function can be evaluated at equal intervals along the *X*-axis. Instead of linear interpolation between successive pairs of *Y*-values, as done in the method of rectangular strips, Simpson's method fits a quadratic curve to each successive set of three *Y*-values. The interval over which integration is required must be divided into an even number of strips, so that the number of *Y*-values will be odd.

My interest in Simpson's method was recently renewed when one of my students needed a table of integrals in order to plot a mathematical model of a meandering river. We wished to define Simpson's method in such a way that it would apply to any monadic function, without recourse to either global variables or loops. Moreover, the result was to be the cumulative integrals over a set of intervals of unequal length.

For illustration here we will use the familiar **n**ormal **dis**tribution function:

```
NDS:(*.5×-⍵*2)÷(○2)*.5
```

and compute the areas under the normal curve from `0` to `.1 .3 .5 .8 1 1.5 2 3` standard deviations (along the *X*-axis).

First we define the **pairwise** function:

```
      PW:⍎2⌽' YX ',⍺,0⍴X←1↓⍵,0⍴Y←¯1↓⍵
```

which here we use for differences

```
      '-' PW 1 2 4 6 8 12
1  2  2  2  4
```

but in which any dyadic function can substitute for the left argument; for example, `'÷'` for pairwise ratios. The use of *PW* for pairwise sums appears in a later illustration. Simpson's method is then defined by three functions:

```
      SDI:(X∘.+(1+⍺)⍴0)+((1↓⍵)-X←¯1↓⍵)∘.×(⍳1+⍺)÷⍺

      XWT:(('-'PW⍵)÷⍺)∘.+(1+⍺)⍴0

      SIMPSON:+\⍵+.×(1⌽1 1,(¯2+0⊥⍴⍵)⍴4 2)÷3
```

Thus:

```
      V
0  0.1  0.3  0.5  0.8  1  1.5  2  3

      SIMPSON (4 XWT V)×NDS 4 SDI V
0.039828  0.11791  0.19146  0.28814  0.34134  0.43319  0.47725  0.49865

      V,[.5]0,SIMPSON (4 XWT V)×NDS 4 SDI V
   0           0
   0.1         0.039828
   0.3         0.11791
   0.5         0.19146
   0.8         0.28814
   1           0.34134
   1.5         0.43319
   2           0.47725
   3           0.49865
```

SDI sub**d**ivides the intervals along the *X*-axis to provide the argument for the function to be integrated; in this case *NDS*. *XWT* provides the **X**-axis **w**eigh**t**s, required because the intervals along *X* are unequal. Finally, *SIMPSON* performs the weightings that are the basis for Simpson's method, and computes the cumulative sums that are wanted.

It should be noted that the three factors controlling the result are clearly visible: the function defining the curve; the intervals along *X*; and the number of strips.

4. Integration of an Unknown Function

A different problem of integration was posed by the need to analyze X-ray fluorescence spectra transmitted from the planet Mars by the Viking landers. The data consisted of discrete values (Y) read in the successive channels (X) of the instruments. It is required to determine the X-value of the center of gravity of the data corresponding to a certain range of channels. This is equivalent to interpolating between channel numbers.

First we integrate by computing the cumulative sum of successive rectangles. Half of the last value is then the center of gravity. To what X-value does it correspond? We determine the indices of the four values adjacent to the center of gravity, and fit a quadratic to the corresponding cumulative integrals. The coefficients of this quadratic are given by:

```
CF:(A[I]⌹(I←(¯1+⍳4)++/A≤H)∘.*0 1 2)-3↑H←.5×¯1↑A←+\.5×'+'PWω
```

The value of `X` that sets this quadratic equal to the center of gravity is found by subtracting the value of the center of gravity from the first coefficient (corresponding to `X*0`) and computing the roots of the quadratic.

```
RQ:((-ω[1])+1 ¯1×((ω[1]*2)-4××/ω[0 2])*.5)÷2×ω[2]
```

Of the two roots, we take the one that lies in the appropriate range, having added 1 to take into account the fact that computed integrals (`A` in `CF`) begin with channel 1 rather than with 0.

Thus the **X**-value of the center of gravity is given by:

```
XCG:((P≤⍴ω)∧P≥0)/P←1+RQ CFω
```

```
      (ιρY),(0,+\.5×'+'PW Y),[.5]Y
     0          0          5.6348
     1          5.5781     5.5215
     2         11.058      5.4381
     3         16.413      5.273
     4         21.588      5.0752
     5         26.722      5.193
     6         31.942      5.247
     7         37.262      5.3936
     8         42.955      5.9915
     9         49.226      6.5511
    10         56.003      7.0031
    11         63.252      7.4955
    12         70.805      7.6109
    13         78.424      7.6256
    14         86.047      7.6207
    15         93.605      7.4955
    16        100.94       7.1701
    17        107.9        6.7452
    18        114.38       6.2146
    19        120.33       5.7038
    20        125.84       5.2983
    21        130.88       4.7875
    22        135.57       4.6052
    23        140.13       4.4998
    24        144.57       4.382
    25        148.92       4.3175
    26        153.16       4.1744
    27        157.3        4.0943

      XCG Y
13.03
```

The reason why we need the center of gravity is that slight distortions of the spectra must be corrected by "stretching". The stretch to be applied is defined when two peaks whose observed and theoretical positions are known. The transformation is of the same type as the change from Fahrenheit to Centigrade, where the high value of 212 becomes 100 and the low value of 32 becomes 0.

```
      STR:ω[0;1]+(α-ω[1;1])×÷/-/ω

      (10×ι11)STR 2 2ρ100 0 212 32
¯17.778  ¯12.222  ¯6.6667  ¯1.1111  4.4444  10  15.556
      21.111  26.667  32.222  37.778
```

5. Blending Analogues of Martian Soil

The X-ray fluorescence spectra transmitted by the two Viking landers enable us to estimate the bulk chemistry of samples picked up from the Martian surface by the instruments' arms. In order to determine the probable mineralogic composition, we must blend terrestrial analogues for testing in equipment identical to that operating on Mars.

LIB is a library of possible materials. *LIB*`[;0]` gives the identification numbers, and the remaining 17 columns give the weight percentages of various chemical elements. *LIB* also includes the compositions of samples analyzed on Mars.

The functions are as follows:

```
      SL:⍉0 1↓α[,α[;0]⍳ω;]

      WT:C÷+/C←ω[;0]⌹0 1↓ω

      TABLE:α,8 2⍕ω[;0],Y,[.5]-ω[;0]-Y←(0 1↓ω)+.×WTω

      DROP:U⍀(U←~(⍳1↑⍴α)∊ω)⌿α
```

SL selects from the library and drops the identification numbers. *WT* computes the normalized weights, or coefficients, for making the first sample from the others. Thus, (with simulated data):

```
      WT DATA←LIB SL 2 1 3 5 7 8 9
¯0.21542  0.42235  0.32931  0.24945  0.53262  ¯0.31831
```

Because negative weights are not possible, samples 1 and 9 cannot be included in the blend to make sample 2.

```
      WT DATA←LIB SL 2 3 5 7 8
0.31034  0.051999  0.19505  0.44261
```

A recursive procedure is obviously called for to select only positive weights:

```
      SPW:αSPW U/ω:(0=⍴ω)∨∧/U←1,0<W←WTαSLω:W,[¯.5]1↓ω

      LIB SPW 2 1 3 5 7 8 9
0.31034   0.051999  0.19505   0.44261
3         5         7         8
```

If *EL* is the appropriate matrix of names of chemical elements, then *TABLE* will tabulate the fit and its residual. (The Viking scientific team determines the residual by subtracting the observed value from the predicted value, rather than the reverse, or common convention.

```
      WT DATA←LIB SL 2,1 0↓LIB SPW 2 1 3 5 7 8 9
0.31034  0.051999  0.19505  0.44261
```

```
      EL TABLE DATA
NA   4.31    5.00    .68
MG   5.56    3.99  ¯1.57
AL   1.64    5.14   3.50
SI   6.54    7.89   1.35
 P   9.42    4.52  ¯4.90
 S   6.57    6.48   ¯.09
 K  10.64    8.55  ¯2.09
CA   6.19    7.12    .93
TI   1.65    2.92   1.27
FE  10.95    9.39  ¯1.56
CL   4.55    4.65    .10
 C   1.57    6.29   4.72
 O   6.28    4.68  ¯1.61
RB   2.81    4.52   1.71
SR   5.44    3.04  ¯2.39
 Y   5.17    7.02   1.85
ZR  10.71    8.80  ¯1.91
```

Because some elements may not have been determined, it is sometimes necessary to drop certain rows from the table. If the chemical symbols are assigned appropriate numeric values, *DROP* will eliminate the elements specified. Thus, to eliminate *P*, *K*, and *SR*:

```
      P,K,SR
4  6  14

      WT DATA←DATA DROP P,K,SR
0.30714  0.097902  0.058802  0.53616

      EL TABLE DATA
NA     4.31    4.52     .20
MG     5.56    3.96   ¯1.60
AL     1.64    5.15    3.51
SI     6.54    9.10    2.56
 P      .00     .00     .00
 S     6.57    6.49    ¯.08
 K      .00     .00     .00
CA     6.19    6.86     .67
TI     1.65    3.17    1.52
FE    10.95    9.39   ¯1.56
CL     4.55    3.97    ¯.58
 C     1.57    5.99    4.42
 O     6.28    4.78   ¯1.50
RB     2.81    4.61    1.80
SR      .00     .00     .00
 Y     5.17    6.77    1.60
ZR    10.71    9.66   ¯1.05
```

It should be noted that no global variables are used in these functions, and that each function is simple. If routine use of the functions would be easier by embedding them in a larger package, which might be conversational or include checks for error, this is, of course, easily done.

Conclusions

According to Berry and others (1971), "when APL functions are executed at an APL system, the user should have to state the mathematical kernel of his function, **and nothing more.**" Direct definition permits just that. Nothing superfluous is required. Indeed on the IBM 5100 even the mechanics of editing a function is trivial: the function is displayed, scrolled to the active line, and modified; execution then causes recompilation.

Direct definition enforces a discipline that makes it far easier to write good APL. It is also a natural way to teach, because functions are introduced (on the blackboard or on the computer) as the subject requires them and without digressions. Expressions that have been developed are simply given names and used. Documentation is straightforward because every concept is a one-line function with no side effects. If the functions have been well chosen and named, it is easy to state what each does. Interactive documentation with appropriate data can illuminate the individual functions and illustrate typical ways in which the functions can interact as a system.

When functions are directly defined, they are likely to be the glass boxes that Paul Berry has urged us to provide.

Acknowledgements

I thank Paul Berry for his invitation to comment on my experience with direct definition of functions. Although my interest in direct definition began with a preliminary version of Iverson's "Elementary Functions", it was greatly stimulated by numerous discussions with Don Orth, who spent a semester as Visiting Professor at Pomona College under a grant from the Mellon Foundation. Allen F. Glazner and Donald Tsusaki made valuable suggestions while working with me on applications of direct definition. I have freely adopted ideas generously provided by Ken Iverson, who (as Sylvester said of his friend Cayley, the founder of matrix algebra) habitually discourses pearls and rubies.

References

1. Berry, P.C., G. Bartoli, C. Dell'Aquila, and V.N. Spadavecchia, **APL and Insight: a strategy for teaching**, Conference on APL, Paris, 1971. Available from APL Press, Pleasantville, N.Y.

2. Iverson, K.E., **Elementary Analysis**, APL Press, Pleasantville, N.Y., 1976.

3. Iverson, K.E., **Programming Style in APL**, APL Users Meeting Proceedings, 1978.

4. McIntyre, D.B., **Introduction to the study of data matrices**, "Models of geologic processes - an introduction to mathematical geology", Editor P. Fenner, American Geological Institute, Washington, D.C., 1969.

5. Orth, D.L., **Calculus in a New Key**, APL Press, Pleasantville, N.Y., 1976.

6. Perlis, A.J., and S. Rugaber, **The APL Idiom List**, Research Report 87, Computer Sciences Department, Yale University, 1977.

GENERALIZED SINGLE PLANT FREIGHT COST MODEL

Lawrence Lapide
Arthur D. Little, Inc.
Boston, Massachusetts

Introduction

This paper describes a generalized APL single-plant freight cost model. The model was developed as part of a plant location study conducted for a major consumer product manufacturer. The model is quite general and can be applied to many industries, as well as to warehouse and/or distribution center location studies.

The model was designed to answer "what if" questions on inbound and outbound shipments and their related costs for any prespecified plant location. Outbound shipments refer to finished goods that are sent from the plant to particular "markets". Markets, for the purpose of this study, were defined as population centroids of individual states. Inbound shipments refer to raw materials or supplies being sent from a set of suppliers to the plant. Supplier locations were defined in the study to be the cities or towns in which the appropriate suppliers resided. All locations were input into the model as latitude and longitude coordinates. The model is capable of computing shipments and costs for any latitude and longitude input as a plant location. The model was capable of performing, in a single run, all calculations for a single plant for as many planning years, markets and suppliers input. It also included both plant capacity and supply capacity as functions of time.

This "what if" type of model was used, instead of an "optimizing" model, since it was anticipated that only a small number of locations would be feasible once all engineering constraints were met. Rather than building a model (e.g., linear programming model) which searched for an optimal location among all conceivable alternatives, this optimal could be found by running the model separately for each of the feasible locations.

The model described here was designed to support a consultant doing an analysis; it needed only to provide results. This is in contrast to computer models which are "turnkey" systems that are used by a client and (1) are usually meant to be used many times (i.e., production-oriented), (2) are usually required to be fully documented and (3) usually possess a detailed "error correcting" prompting structure to guide the user. Turnkey systems usually require longer lengths of time to develop. This plant location model, being of the consulting analysis type, enabled the designer to develop a prototype of the model in several days. One reason for this quick development time had to do with the fact that no new concepts were required; once a conceptual flowgraph (to be described in the next section) of the calculations was developed, APL's higher level, matrix-oriented features allowed the consultant to program the system very quickly. This quick development time was found to be of great advantage to the study, since it was possible to present an early on-line demonstration of the prototype model to the client. This gave him a quick understanding of the concepts involved and also gave him incentive to start collecting the model data required, since he understood the uses of all the data. The interactive nature of the APL language made this on-line demonstration possible.

The remainder of this paper discusses the model itself in terms of (1) the concepts embodied in the model, (2) the use of a flowgraph to design it, (3) the detailed description of functions, and (4) the model's outputs and their uses. The paper also contains a final section describing that a model of this type possesses desirable attributes of a management decision model. The use of APL facilitated the implementation of these attributes for this model.

Description of the Model

Concepts Embodied in the Model

The model was constructed using standard procedures for computing freight costs incurred from a particular plant location. The model was conceptually constructed using a "top-down" approach in which elements and factors are included, as needed, to compute total freight costs. The first breakdown of total costs is, for example, outbound versus inbound freight costs. The procedures used for each of these elements are described as follows.

1. Outbound Freight Costs

In order to estimate outbound freight costs, quantities of shipments made to each market must be known, as well as the unit costs involved in shipping to each market. To determine the former, programming logic on which markets will be serviced first in cases where limited plant capacity makes service to all markets impossible, must be considered. At least two types of decision rules are reasonable. The first type of rule is that the plant will first service those markets which have the lowest per unit shipping costs. Under usual assumptions, this basically translates to "closest markets get serviced first". The second type of decision rule is marketing-oriented and involves first servicing those markets that are consistent with a pre-determined marketing strategy. This might translate to programming logic which uses a ranked market approach to determine which market will be served when capacity becomes limited. Each of these decision rules are realistic and reasonable, as well as being easily implemented into an APL computer model. For the study described here, the client did not have a particular market strategy; hence a least-cost or distance decision rule was applied.

In order to determine the unit costs for each outbound shipment, a relationship between distance and per unit costs for each transport mode is necessary, unless of course, a table can be generated and looked up within the computer model. For an arbitrary plant location, a general table would be excessively large. To compute distance between an origin-destination pair of locations, the "great circle" formula for computing distance between two locations based on their latitude and longitude. Distances computed by this formula were modified by a factor which has been developed in a series of studies and relates straight line distance to road mileage. To relate unit shipping costs to distance, a relationship between these was developed using historical unit costs for a set of sampled origin-destination pairs and their corresponding distances (linear regression was used). Separate models were developed for each transport mode. Model decisions were programmed by using logic which depended upon distances; that is, destinations greater than a specified distance would be sent by rail, while the others would be sent by truck.

2. Inbound Freight Costs

As with outbound freight costs, inbound freight costs require estimates on quantities of shipments being made from each supplier and the per unit cost of shipping from each. Raw material requirements and a decision rule for choosing between suppliers is needed. The former are easily handled by obtaining a requirements list which relates how much of each type of supply is required to produce each unit of output for each type of packaging. Once obtained, the model easily computes supply requirements by applying the list against the plant's product mix output.

Although one can conceive of several decision rules to choose suppliers, the most reasonable is the least-cost decision rule, which was used in this model. Those suppliers having the lowest per unit shipping costs (i.e., closest to the plant) would be used first until their capacity limit was reached.

The per unit inbound shipping costs for suppliers were obtained in the same way as the outbound shipping costs; however, separate historical relationships between per unit costs and distance were developed for each type of raw material.

Flowgraph of the Model

Figures 1 and 2 represent flowgraphs of the general calculation procedures involved in the model. Each broad-rimmed box represents an APL function which produces an output, while each thin-rimmed box represents a data array which is used by appropriate functions. The diagram has been split into two parts for purposes of clarity; the original was developed with both parts integrated.

Before describing the details of the calculation flow, several points concerning these types of flowgraphs should be mentioned. The first point is that while these types of flowgraphs are conceptually simple, they have been found to be extremely useful for the structural design of APL coded computer-based systems. Once developed, the coding is usually fairly straightforward. (This should be evident from the detailed model description in the next section.) The graph displays the fact that the functions, or programs, are merely manipulating data elements - usually matrices or vectors - to produce outputs. These outputs are then passed on to other programs which do the same.

A second point to note about these flowgraphs is that they can be easily explained to a person who knows little or no APL and can also be developed by that same person. In a consulting environment, this is useful because a client without knowledge of APL should be able to easily understand and contribute to the model structure.

The last point to be made is that the flowgraph aids in developing submodular designs. These designs are useful for two reasons, the first of which is that intermediate results can be easily displayed. This allows one to demonstrate the model on-line by showing the outputs of each function in sequential order. This enables the person being shown to follow the flow by actually seeing the outputs. This type of graph, in conjunction with an on-line demonstration on a terminal, has proved effective in communicating APL models to various clients. The second reason is that each submodule can evolve by allowing the designer to start with a simple function and revise it as study progresses and more complex manipulations may be required. For example, in the model described here, one could start with the least-cost market decision rule and at a later time revise it to incorporate a marketing strategy rule.

Detailed Description of Functions

This section contains descriptions of each of the functions shown in Figures 1 and 2 in terms of the inputs and outputs (i.e., vectors and matrices) of each function. Once the translation of inputs to outputs is described, codes for each program are quite easily inferred or developed. The description of functions for the outbound freight calculations are described first. (Please refer to Figure 1, where necessary.)

1. *DISTANCE* - this function is dyadic and takes a vector and matrix argument. The vector argument has two elements and represents the latitude and longitude of the plant location being run. The matrix argument is a two-column matrix, which represents markets along the rows, and latitude and longitude along the columns. The function produces a vector of distances from the plant to the markets as an output. The function is a general routine which is also used to compute distances from suppliers to the plant in an equivalent manner.

2. *DEMAND* - this function produces a matrix output which represents the total industry demand for the product; rows represent markets and columns represent years in the planning period. Two data elements used in the function are matrices which represent per capita consumption and population for each market and planning year.

3. *SHARE* - this program produces a matrix output which represents the potential sales that can be achieved by the plant, if it penetrates the corresponding markets. Markets are represented along the rows, while planning years are represented along the columns. The function calls the *DEMAND* program and uses a data element which is a matrix of market share assumptions for each market and each year.

4. *SALES* - this program is monadic and takes the plant location vector as its argument. The output is a matrix which represents sales made to each market for each year. To determine which of the potential markets will be served with the available plant capacity, the function uses (1) the least-cost decision rule, (2) the output from *SHARE*, (3) a vector of plant capacities by year and (4) the output from *DEMAND*.

5. *OUTFRT* - this program is monadic and takes the plant location vector as its argument. The output is a matrix which represents the total cost of shipping to each market and which has the same dimensions as the *SALES* function's output. The function uses (1) the output from *SALES*, (2) the output from *DISTANCE* and (3) the unit freight cost parameters (two vectors representing linear regression parameters for all outbound and inbound type shipments).

The outputs resulting from calling the *DEMAND*, *SHARE*, *SALES*, or *OUTFRT* functions are equivalently dimensioned matrices. By calling each separately in sequence, various intermediate results can be traced in order to (1) debug the program, (2) understand its flow, and (3) see all outbound implications of a plant's location.

The description of functions for the inbound freight-calculations are described below (please refer to Figure 2 where necessary).

6. *SUPPLYMIX* - this function is monadic and takes the plant location vector as its argument. It produces a matrix of supply requirements which represents types of requirements along the rows and planning years along the columns. To produce its output, it uses (1) the output from *SALES*, (2) the packaging mixes (a 3-dimensional matrix of package percentages for each package type, market and year) and (3) the list of requirements for each unit sold (a 2-dimensional matrix of requirements for each type of supply and package type).

7. *SUPPLIERS* - this is a monadic function which also takes the plant location vector as its input argument. The function produces an output matrix of requirements drawn from each supplier for each year. Suppliers are represented along the output's rows and planning years along its columns. To produce its output, the functions uses (1) the inbound least-cost decision rule, (2) the output from *SUPPLYMIX*, (3) a cross reference suppliers-to-supply type matrix, (4) the supplier distance vector computed by the *DISTANCE* function and (5) the supplier capacities matrix (by supplier and year). (A diagnostic routine is imbedded in the function to pinpoint cases where total supply requirements exceed supply capacities.)

8. *INFRT* - this is a monadic function which takes the plant location vector as its input arguments. It has a matrix output which represents the total costs of shipping from each supplier for each year and which has the same dimensions as the output from the *SUPPLIERS* function. To produce its output, it uses (1) the output from the *SUPPLIERS* function, (2) the supplier distance vector produced from the *DISTANCE* function and (3) the unit freight cost (linear regression) parameters.

Similar to the outbound functions, the outputs resulting from calling *SUPPLIERS* and *INFRT* are equivalently dimensioned matrices. By calling each in sequence, as well as the *SUPPLYMIX* function, various intermediate results can be traced to help debug and follow the flow.

Model Outputs and Their Use

As can be noted from the above detailed descriptions, all functions have outputs which have one of the three following dimensions: (1) markets along the rows and years along the columns, (2) supply type along the rows and years along the columns, or (3) suppliers along the rows and years along the columns. Because of this, a generalized reporting function was easily written which was able to take any output desired and format it with the appropriate headings along the columns and labels along the rows. This function enabled the user of the model to produce the following set of outputs for each plant location input.

1. Total industry sales by market and year;
2. Potential plant sales by market and year (i.e., unlimited plant capacity);
3. Actual plant sales under limited plant capacity by market and year;
4. Supply requirements by type and year;
5. Requirements supplied by each supplier for each year;
6. Outbound freight costs by market and year; and
7. Inbound freight costs for each supplier for each year.

These outputs are potentially useful for supplying at least three functional parts of a company with information concerning the implications of locating a plant at a specified location. Cost-related outputs are directly useful to financial management personnel in making business investment decisions. Shipments outputs are directly useful to traffic and transportation management personnel in making scheduling, carrier and model decisions. Finally, supply requirement outputs are directly useful to purchasing management personnel in making decisions on vendors and long-term vendor agreements.

Model Attributes

This plant location model possesses all qualifications which John D.C. Little of M.I.T. attributes to a good decision calculus. That is, this management decision model is (1) simple, (2) robust, (3) easy to control, (4) adaptive, (5) complete on important issues and (6) easy to communicate with. Several of these are inherent in the model because of the way in which it was designed; more importantly, APL facilitated their implementation. These qualifications are described below.

Robust - The model will always give reasonable answers, since it is based on realistic assumptions.

Easy to Control - The user knows how to make the model work and behave as required. In this case, the submodular flow, easily implemented in APL, allows the client and consultant to understand all results.

Adaptive - The model can be changed as required, since it involves a simple change of a function, or a data element, both easily done interactively with APL.

Complete on Important Issues - This attribute usually conflicts with simplicity, but in this location model, the simple assumptions incorporate most elements needed for computing freight costs.

Easy to Communicate With - The model in this case can be easily changed and rerun since APL is an interactive language.

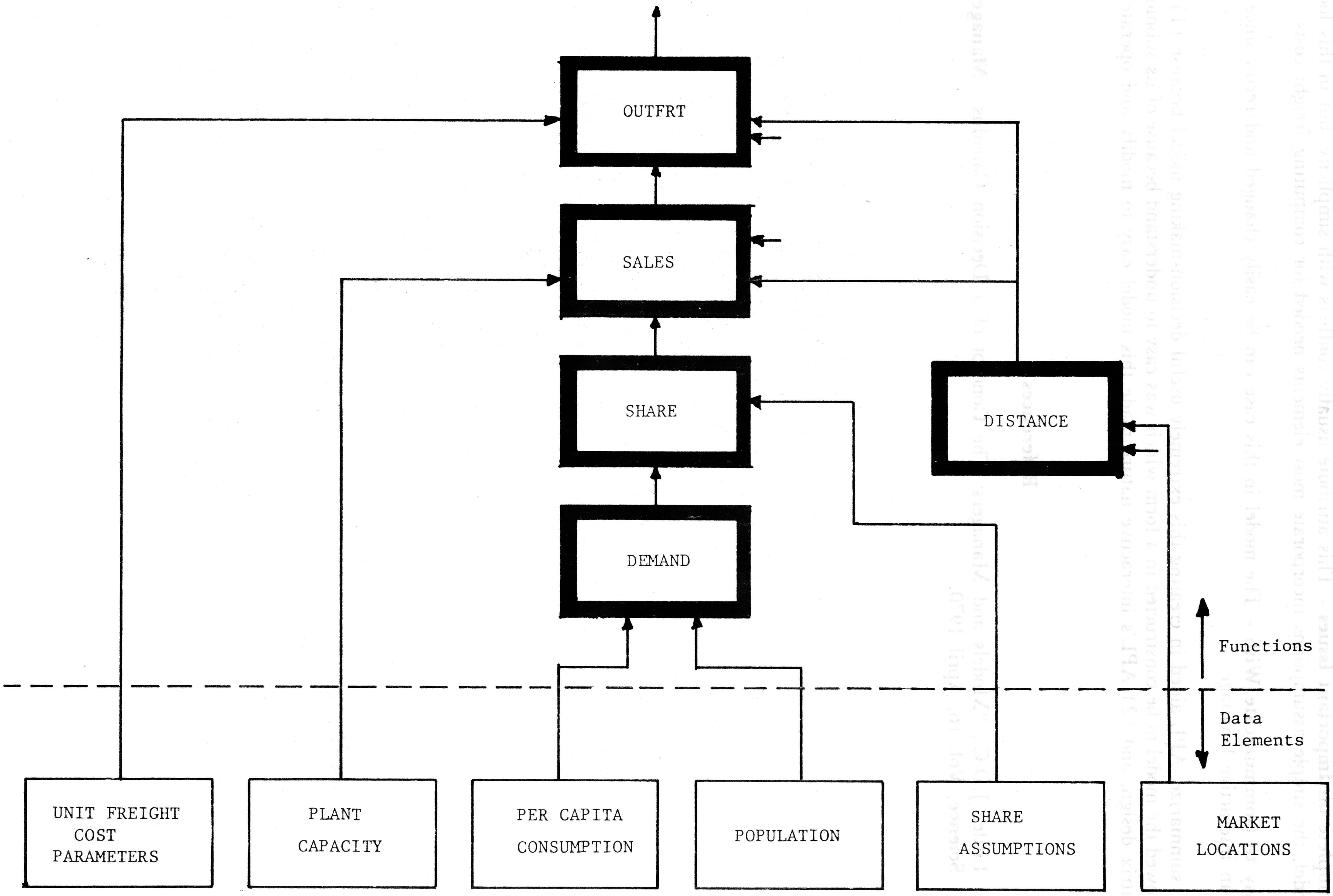
To summarize, APL aided in creating this extremely useful decision-making model because (1) APL allowed the model to be constructed in a form which was easy to understand because of its submodular matrix design, and (2) APL's interactive nature made this model easy to modify and operate.

References

1. Little, J.D.C., "Models and Managers: The Concept of a Decision Calculus", **Management Science**, Vol. 16, April 1970.

FIGURE 1

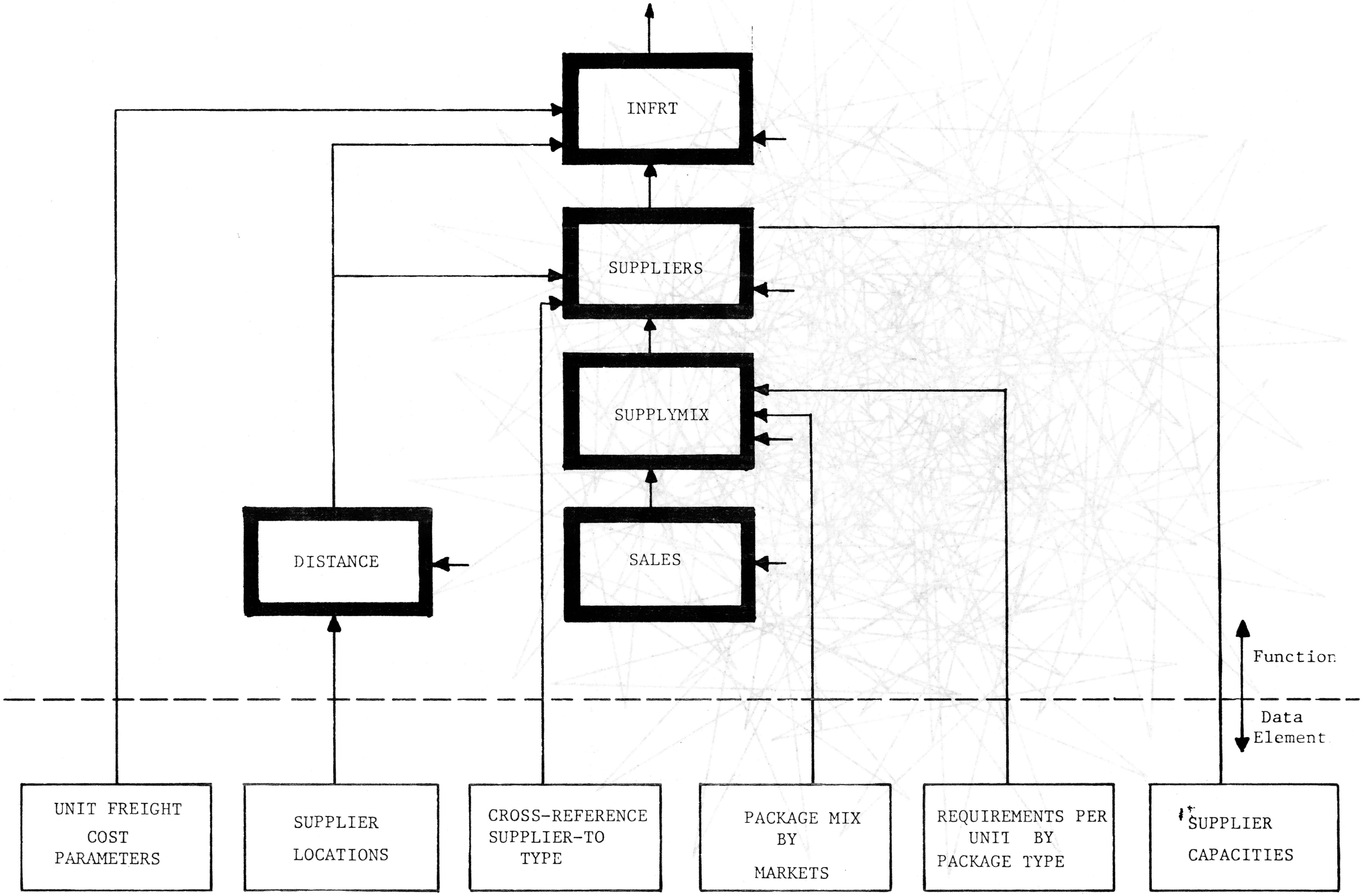
GRAPH OF PROGRAM FLOW FOR OUTBOUND FREIGHT COSTS*


OUTFRT
SALES
SHARE
DEMAND
DISTANCE
Functions
Data
Elements
UNIT FREIGHT COST PARAMETERS
PLANT CAPACITY
PER CAPITA CONSUMPTION
POPULATION
SHARE ASSUMPTIONS
MARKET LOCATIONS

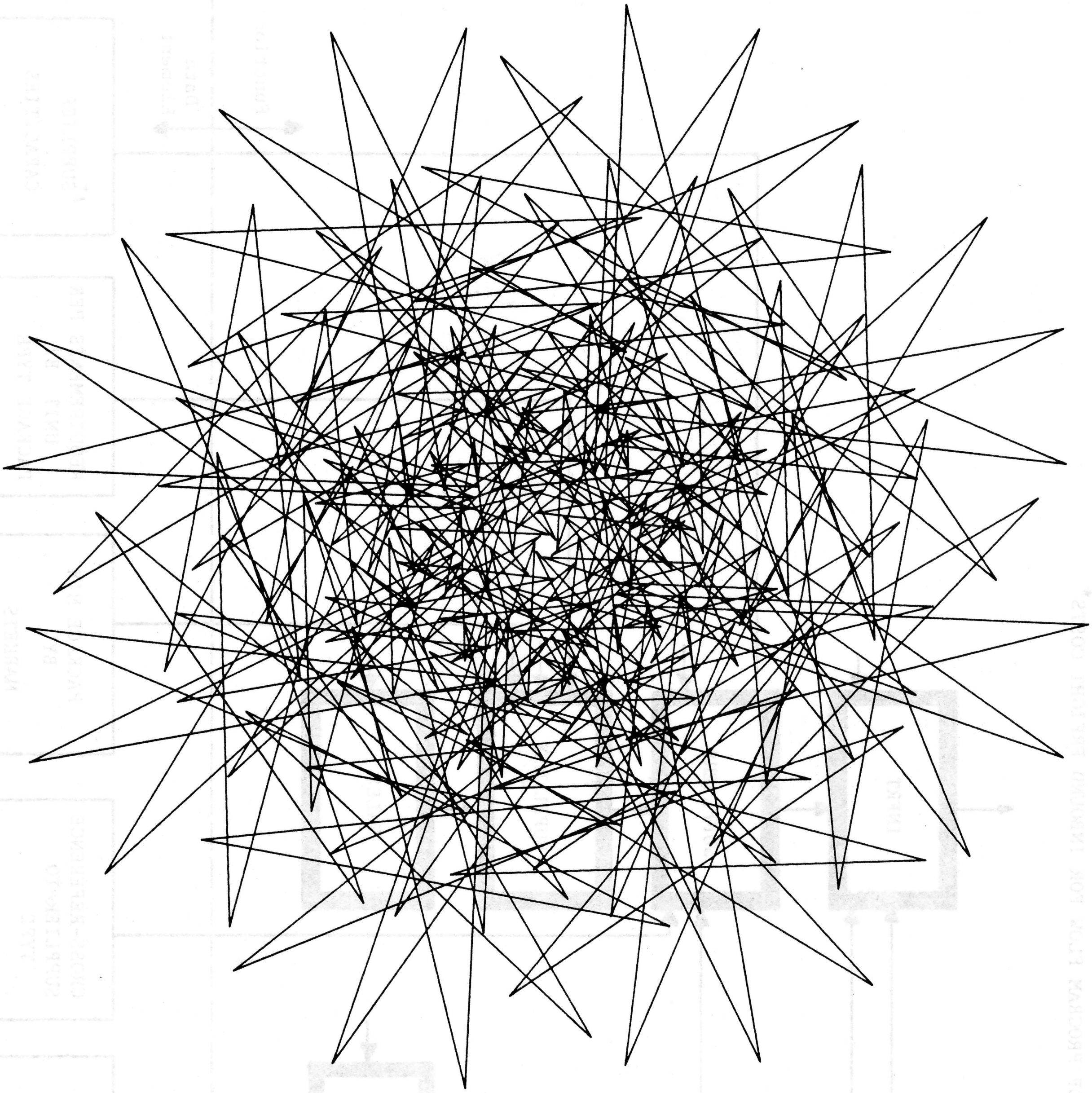
*← Refers to plant location vector as an input argument.

FIGURE 2

GRAPH OF PROGRAM FLOW FOR INBOUND FREIGHT COSTS*


INFRT
SUPPLIERS
SUPPLYMIX
SALES
DISTANCE
Function
Data Element
UNIT FREIGHT COST PARAMETERS
SUPPLIER LOCATIONS
CROSS-REFERENCE SUPPLIER-TO TYPE
PACKAGE MIX BY MARKETS
REQUIREMENTS PER UNIT BY PACKAGE TYPE
SUPPLIER CAPACITIES

* ← Refers to plant location vector as an input argument.



APL PROGRAMMING IN A DYNAMIC PLANNING ENVIRONMENT

Gregory Koutsouros
Xerox Corporation
Rochester, New York

Abstract

The paper will discuss the roles of, and the relationship between, the corporate planner and the program consultant during the various phases of design and implementation of a computing project. The project will be developed to support the work of a planning department characterized by rapidly changing requirements and end users who are sophisticated in their own work but are not generally sophisticated in programming. Emphasis will be placed on documentation as a way of developing and programming a system and the interaction required between the above individuals to assure adequate technical documentation. The end objective is to provide a set of guidelines for documentation which can be used during the development of a computing project. Hopefully, these guidelines will help to bridge the gap between the analyst, inexperienced in programming, but sophisticated as a corporate planner, and the programming consultant, who is sophisticated in programming, but inexperienced with the needs of a dynamic planning environment.

Introduction

APL is an extremely powerful analytical tool. When applied to a business planning environment, APL enables the analyst to effectively, and in a timely manner, pursue solutions to problems which might otherwise lack the rigor of appropriate analysis. Without such rigorous and timely analysis, suboptimal decisions can be made impacting the business. In the photographic market, technology has concentrated on improving methods so that the photographer can be free to focus on the desired resultant picture. And, in the same sense, there is a need to improve current methods of programming so that the analyst can be more free to concentrate on the analysis.

The planning environment we are about to discuss is characterized by the need for quick turnaround to meet day-to-day decision deadlines to operate the business. In such a business where profit objectives and lead times require dynamic changes in operations, software requirements will often require modifications to existing methodologies employed in an already constructed APL system. Because time-sharing is often the vehicle employed in this environment, and the analysts are sophisticated in their own work but not generally sophisticated in programming, the ability and ease of maintaining, modifying and enhancing software becomes a crucial factor in the analyst's ability to react and quickly respond to a business question.

The planning environment usually consists of a group of users who buy time or share the use of a computer, and therefore, are not concerned with the design of the computer hardware. The hardware costs incurred by this group are absorbed through usage fees, whether they be actual payments to a vendor or simply intracompany transfer fees. In addition to these usage or operating expenses, the remaining costs of using the computer are incurred by the resource required to design, develop, maintain and enhance the system software. Operational and developmental costs have long been the thrust for controlling costs. However, maintenance and enhancement costs often go without consideration in the implementation of a system. These costs are downstream actions which often occur in a piecemeal manner so that the true time and costs remain invisible to the user. Even if the designers tried to anticipate such costs in the development of a project, it would be doubtful they could foresee the true expense that would accrue. There is a way, however, that these unanticipated costs can be minimized, and that is through appropriate levels of functional documentation.

In summary, the ability to enhance, maintain and modify software impacts the business planning environment in two ways -- the ability of an analyst to respond to a business question is improved, enabling management to make better and quicker business decisions in a dynamic environment; and the cost of operating a system will be minimized and its life-cycle extended by reducing the resource required to maintain and enhance the system. For these reasons, functional APL documentation will be explored in this discussion as a necessary ingredient for efficient use of APL systems in a dynamic environment by providing an efficient framework to enhance, maintain and modify APL software.

Overview

In the remaining paragraphs we will investigate some ideas for documenting APL systems as the key ingredient in providing the ability to enhance, maintain and modify software. We will develop the relationship between the corporate planning analyst and the programming consultant during four of the seven phases of a computing project. A computing project can be divided into seven phases with characteristics as shown below:

Design: Identify and define system requirements

Development: Refine system requirements into function specifications

Coding: Refining function specifications into APL code

Testing: Validating system operations

Final Documentation: Integrating system user and technical documentation into a functional reference

Maintenance: Correcting system operations which are thought to have been validated during testing

Enhancement: Adding new capabilities to the system or revising current capabilities to achieve efficiency in operation

The four phases that we will explicitly explore in the discussion, in reference to guidelines for documentation, are the design, development, coding and final documentation stages. Emphasis will be placed on documentation as a way of developing and programming a system, and the interaction required between the above individuals to assure adequate functional documentation. The end objective is to provide a set of guidelines for documentation which can be used during the development of a computing project. Hopefully, these guidelines will help to bridge the gap between the analyst, inexperienced in programming but sophisticated as a corporate planner, and the programming consultant, who is sophisticated in programming but inexperienced with the needs of a dynamic planning environment (these guidelines should appear equally useful to the analyst who has to pursue the development of a system without the expertise of a programming consultant). If well conceived from the onset of a new system and pursued by the analyst and consultant with consistency, appropriate APL functional documentation enables the analyst to be more free to concentrate on the analysis supporting the business decision to be made.

There are two types of documentation which provide direction in how a system operates. User documentation is oriented towards the user describing the functional interaction required to operate the system to perform the intended analytics. Technical documentation focuses on the technical programming involved in the system describing the interactive functions, process, and data storage mechanisms used in the system. There is a fine, often vague line of differentiation between these two types of documentation. As we look into the purpose for each type of documentation, however, we see that there is a major difference. User documentation helps the analyst operate the system, as it currently exists, and produce analytical results on a day-to-day basis by describing conceptual purpose, data setup, prompts, progression of function use, options, et cetera. This documentation is not intended to facilitate modifications, enhancements or maintenance of the system. Technical documentation, on the other hand, serves the purpose to facilitate understanding of the code so that the system may easily be modified, enhanced and maintained.

Technical documentation is not mutually exclusive from user documentation. The individuals designated to maintain the system will improve their understanding of the system in total by understanding the purpose and operational use of the various functions employed in the system through documentation specifically designed for the user. The remaining discussion primarily focuses upon technical documentation and the purpose it serves. Although we will not specifically discuss developing forms of user documentation, we still recognize its importance in enabling the analyst to efficiently operate the system to respond quickly to a business question. Development of specific user documentation techniques is left as a separate exercise, but there will no doubt be extensions of the applications discussed which also apply to user documentation.

Design

The design of the system requirements is primarily the responsibility of the analyst. This responsibility does not preclude the necessity to establish and maintain open lines of communication and respect towards the programming consultant during the design phase. In fact, the analyst will need the assistance of the programming consultant to integrate the conceptual and analytical needs of the system with the available APL system facilities. But, it is the analyst's responsibility to extract a set of process requirements from the business environment to meet the analytical needs of the corporation.

The first consideration for the analyst to recognize during the design phase is that **documentation is a form of developing a system**. System development is usually a shared responsibility between the corporate planner and a programming consultant. Communication between these individuals must be concise and clearly understood by both parties to achieve results that are consistent, easily maintainable and enhanceable. Initial documentation of system requirements can become a framework which provides clear and concise communication resulting in the development of the project into an easily maintainable and enhanceable system. And as a by-product, much of the information required to document the system after programming and testing is completed becomes available through the evolution of this process.

The second, but probably most important consideration, is that **documentation is a form of programming**. The way you structure your functions as well as the way each function is coded can, and should, be your first echelon of technical documentation. This implies that documentation standards must be conceived and pursued from the beginning of a project. This point is so important that we will restate it as Rule 1:

RULE 1: Proper documentation is conceived as part of the project prior to any coding.

For many expert as well as novice programmers, documentation is something that is not even considered until the coding of a system is in its completion. At that point an ex post facto discussion is often provided. Such documentation, however useful, does not facilitate maintenance and enhancement actions, and therefore, does not meet the objective of reducing resource time and cost for system maintenance, enhancement and modification. Without the facility to react quickly, the analyst often confronts the situation of letting management make a "gut-feel" decision because the decision deadline is rapidly approaching. This reaction defeats the purpose for providing analytical information to reduce the risk inherent in management decision-making.

Documentation is a form of developing and programming a system and must be initiated by satisfying Rule 1. This rule can be satisfied by recognizing that the need for documentation is in itself a system requirement. To recognize these objectives, there must be agreed upon guidelines describing how documentation will be utilized to develop and program the system. In summary, we restate this important need as Rule 2:

RULE 2: Documentation should be formally recognized as a system requirement.

This rather obvious statement is too general to suggest to the analyst and programming consultant what specific actions will have to be taken to meet such a requirement. Later on we will develop guidelines for documentation which can be used initially as a format for designing the actual user and analytical requirements. And if properly pursued, these written system requirements, during development and coding phases, will evolve into the second echelon of system documentation -- written documentation.

Investigating the concept that documentation is a form of developing and programming a system further reveals that a system, properly structured and programmed with consistency and flexibility, will be, to some degree, "self-documenting". Thus follows Rule 3:

RULE 3: Documentation is programmed into your system through structure, consistency and flexibility.

With the addition of a few well placed comments interspersed throughout the code, a majority of the functions become substantially "self-documenting". The remaining written documentation is then easily developed around a well designed system and serves to supplement this "on-line" documentation to facilitate maintenance and modification.

Structure

Structuring the system into component modules is the first step in developing a well documented system. Each module alone can be characterized as a black box with a prescribed input, a process, and an output as shown in Exhibit 1. Essentially, from an extreme point of view, each APL statement can be classified as one of these black box modules with input, a process and an output. However, module definition should be more directly associated with a significant analytical step in the system. Ultimately, system requirements can be refined into three types of functions -- control modules, process modules, and utility functions.

Development of the system into component modules is an interactive process between the analyst and consultant which starts by defining each user or system requirement. Exhibit 2 generically depicts a typical APL system application. The analytical needs of the business as determined by the analyst are shown as data input, data analysis and data output tasks. We will refer to these user needs as tasks (not to be confused with T-task, N-task and B-task terminology). Each task can, from the analysts point of view, be associated with an integral part of the analysis. Distinction of the user requirements as tasks starts the design process. Eventually, these tasks will be turned into control, process and utility functions.

Referring back to the black box description, we can now begin to determine task documentation guidelines which help us develop the system. Each task is similar to the black box module as it requires some input, perfoms data processing, and provides output. Therefore, the analyst begins the conceptual design of the system by defining the tasks required, and for each task, its inputs, processes and outputs. Exhibit 3 expands the black box application to task definition and in Exhibit 4 we begin to develop an illustrative example by employing this technique. In this example, because the requirements are fairly simple, the task definition is quite detailed. The example still requires refinements, but in practice, the first set of task definitions are usually more general in reference to the size, shape, and data base structure. Refinement of these definitions then continues to occur through the interaction between the analyst and the consultant.

Review of the example reveals some more detail to the structure of task definition. The input variable list should include a description of the variables, a suggested variable name for each, and a prescribed structure with identification of the various dimensions. In addition, the source for each variable should be identified.

Process definition should describe the system application being performed. Referring back to Exhibits 2 and 3 we recall three classifications of system applications -- data input, data analysis, and data output. The example shown in Exhibit 4 illustrates each of these classifications in the two identified tasks. Task 1 is an input application and Task 2 is a combination of the analysis and data output classifications. The system application will always be any one or a combination of these categoric descriptions and should be understandable from a user oriented perspective, including a detailed definition of the process or algorithms that are pertinent. Additional delineation should be focused at user conventions and options that are required by defining user prompts, task conventions (T-task, N-task, and/or B-task options, terminal and/or highspeed printing options, et cetera), input data validation checks and report formats. Remaining descriptions should identify intermediate variables which are neither input or output results.

Output data should be defined including variable descriptions, suggested names for each, and a prescribed structure with an explanation of the various dimensions. In addition, the destination of each variable (i.e., where it is going to be stored or printed) should be identified. Exhibit 5 summarizes each of the items presented above by expanding upon the task definition as presented in Exhibit 3.

From a user or analyst perspective, the variables defined during task definition will be primarily conceptual in nature in comparison to what we will refer to as working variables. A clarification of these terms follows:

(1) Conceptual variable: those variables which have a conceptual meaning to the user in the analytical process such as a mean, standard deviation or sample-size in a One Way Analysis of Variance, or variables which represent the description of a system entity such as an exercise or report number.

(2) Working variable: an extension, combination or permutation of one or more conceptual variables which are needed from an APL perspective for temporary manipulation of data in the analytical process such as counters, indices, limits, variables for sorting and temporary storage to reconfigure matrices for proper addition, subtraction, multiplication, division, et cetera.

The reason for identifying these distinctions is that initial task definition, which is primarily an analyst responsibility, primarily identifies conceptual variables. Not until the program consultant begins the function definition phase will most of the working variables be identified. Therefore, during the design phase, the analyst has the responsibility of designating the conceptual variables which will be required to achieve the system application, recognizing that working variables will be designated at a later point by the programming consultant. The analyst should select descriptive names that conceptually convey to other analysts, who might use the system, what the variable represents. An ideal way to present this information is in an alphabetical glossary of variables. The important consideration is to document these variables so that they are most easily recognizable at a later time when trying to enhance and maintain the system.

Once the tasks are identified and defined, the analyst and the consultant can design the data base structure. In APL there are three sources and three destinations of data entering and leaving the system:

Sources

(1) Data entered through quad and quad-quote user input

(2) Data previously stored in a variable within the workspace

(3) Data stored in file

Destinations

(1) Store in file

(2) Store in global workspace variable

(3) Printed and discarded, or simply discarded

The first step in defining the data base is to identify the source and destination of variables into the above classifications. By reviewing the input and output variables identified during task definition, a list of variables whose data will be stored in an APL file can be cumulated. Once file variables are identified, workspace variables can be defined (with the advent of the ⎕*PACK* system utility, one should consider whether any global variables should be stored in the workspace -- instead, global variables can be defined through a latent expression alleviating the problem of global variables mistakenly being erased from the master workspace). And the remaining variables are either quad or quad-quote input or are printed and discarded output.

Designing the data base file structure is never a simple task because one is never certain of the end form of the system. Therefore, contingencies should be considered by leaving room for growth in the file structure. As discussed above, after task definition, conceptual variables are available, but working variables are probably not yet defined. Therefore, the potential exists for storing additional data in a file for which variables are not yet defined. Empty components should be retained during design to provide for this contingency.

There are two general file structures which can be developed with each having a different impact on the ease of maintaining and enhancing the system. The two formats are fixed format and floating format file structure. The fixed format file structure is usually easiest to document and easiest to utilize when "debugging" or maintaining the system. The floating format file, while being more complex during debugging or during maintenance is more flexible and easier to enhance. In addition, appropriate APL utility functions can be developed to assist and ease the interaction with the floating format file during system maintenance. The format concepts are reviewed in Exhibit 6 in more detail. Exhibit 7 displays decision criteria for determining whether or not a fixed or floating format should be developed placing preference on the floating format. Except in the simplest of systems, or where no enhancements are going to be entertained (which is not characteristic of a dynamic planning environment), the floating format structure is recommended. As shown in Exhibit 7, the potential costs of maintaining and documenting the floating format file are potentially covered by the avoidance of time consuming revisions to APL programs because of fixed format component revisions and the associated cost of converting existing fixed format files to the new structure.

Data base file structuring should commence with an extraction of the conceptual variables, and any working variables which were identified during task definition that are going to be stored in file. Such a list is shown in Exhibit 8 for the illustrative example. From this list variables can be classed into (1) those which are common to all exercises or contain data for each exercise (or each report) and (2) those which are specific to an exercise (or report). The analyst and consultant then designate the file format to be used and provide initial File Definition as outlined in Exhibit 9 and illustrated using the example case in Exhibit 10. As shown in Exhibit, 10 the floating file format was selected assuming the environment is dynamic with the application and frequency of use potentially expanding. The variables which are common to all exercises are then stored in the fixed portion of the floating format file, while those variables which are for a specific exercise are stored in the variable portion of the file as indicated in Exhibit 10.

The remaining step required to complete the initial design phase is identifying the workspace interactions required and the tasks which are pertinent to each. A Design Workspace Definition is presented in Exhibit 11. With the advent of packed functions (⎕*PACK*) and canonical representation (⎕*FD*), the various types of workspaces required can be conveniently limited depending upon the interaction control that the analyst desires. In the limited case of the example, there need be only one workspace. If the N-task or B-task facility were to be used, a utility workspace for each would be required. In addition, when various applications and levels of security are required, multiple user workspaces should be designated.

Consistency and Flexibility

Just as a good structured design of the system is a form of programming, so is maintaining consistency and flexibility in the system. Consistency means to always code similar processes using the same conventions. This technique makes programs very readable as the individual inspecting the code will not have to decipher every loop, and every branch, et cetera, to determine what action occurs next. Flexibility means that the program is designed with the intention that system constants will be eventually changed. In a practical sense, one should never assume a process or value of a variable to be rigid unless restricted by cost.

Standards for programming conventions which employ consistency and flexibility as an objective have been provided by many APL enthusiasts. The choice of these conventions should be left to the approval of the analyst, primarily because it is the analyst community which must provide the first level of maintenance and enhancement capabilities in the dynamic planning environment. These programming conventions should be included as a system requirement as part of the documentation standards guidelines, developed and recommended by the programming consultant (as the consultant probably has preferential standards) and agreed upon by the analyst. A sample list of conventions which we have found useful are displayed in Exhibit 12. This list is neither exhaustive or mandatory; however, it does emphasize some of the key areas that should be coded with consistency throughout the system to increase the readability and understanding of the code. The important point to remember is that regardless of the mechanisms employed, the mechanisms should be applied consistently.

Design Phase Summary

The design phase is the primary responsibility of the analyst who must coordinate and direct the computing project in a positive, business-like manner. The major objective of the design phase is to define the system requirements in a manner which will expedite the translation of the requirements into system specifications and APL coding. The result will be an integral, well-structured analytical application. The thrust of the analyst's definition of system requirements should not, however, be limited in scope by defining only analytical requirements of the system. Rather, the scope of system requirement definition should set a framework for smooth development of the project with emphasis on building an easily maintainable and enhanceable system through documentation. The design of the user requirements are expedited by employing the vehicle of definition formats and guidelines which produce a well structured and documented system. And as a by-product, written system documentation is readily available to be integrated with on-line documentation into an intensive reference tool. A summary of the design steps is provided in Exhibit 13.

Development and Coding Phases

Commencing the development phase of the system signifies progression of the project from a conceptual design stage to the construction of the technical specifications required to code the system. From this perspective, the groundrules which are founded during the design phase become the framework for adding the detail which permits system implementation. These sets of details are really system specifications which can then be turned into APL code. As the analyst is the expert in the conceptual needs of the system, and therefore, maintains primary responsibilty for design of the requirements, the programming consultant is the technical expert and maintains primary responsibility for development and coding of the system specifications.

The thrust of this discussion is to develop the relationship between the analyst and the programming consultant and show how such a relationship can lead to a well structured and documented system which is easy to enhance and maintain. Since the program consultant is sophisticated in programming and is primarily responsible for the development and coding phases of the project, we are not going to suggest how the system specifications should be developed. Rather, it is our concern that system specifications are developed before the coding phase, and at least informally documented and reviewed by the planning analyst. Therefore, we will suggest documentation formats for system specifications which will help the analyst and consultant communicate during the second formal iteration of the construction of the system.

Development of the system specifications is a refinement of the task definitions identified during the design phase into control, process and utility functions. Expansion of the task definition into these components also provides more detail which enables the enhancement of workspace and file definitions.

The communication vehicles by which the programming consultant informs the analyst are through diagramatic task structures and function definitions. No doubt the consultant will have to devise these items to define the system specifications whether it is done formally or informally using flow diagrams, code listings (off-line), or combinations of the two. The important point is that the process is pursued off-line and that no actual coding (except to test an idea) is performed until the system definition is complete for all tasks. There are three good reasons for proceeding in this way. First, while designing one function, the consultant may want or need to redesign another part of the system. Second, review of the system specifications by the analyst may point out discrepancies between what the analyst expected and what the consultant understood. And third, the consultant can point out requirements which are incurring large amounts of development time or which will require large amounts of operatng expenses. The analyst may then be able to revise the requirements or algorithms so that these costs can be reduced. If the functions had been coded on-line, the consultant and analyst would find it more difficult (at least psychologically) to initiate the change. By iteratively reviewing the system specifications, the analyst and the consultant can agree upon the appropriate requirements before the applications are coded on-line.

Diagramming task structures provides an efficient framework for understanding the operation of the components which comprise a task. Exhibit 14 shows a generic task structure design. The structure is segmented into control, process and utility functions. The generic example shows three levels of functions excluding the utility functions. Most tasks should be subdivided into no more than three levels of subordinate functions to optimize the understanding and maintenance of a task. However, there are applications where more levels are appropriate such as when canonical representation (⎕*FD*) or packed functions (⎕*PACK*) are used. It is suggested that as few levels of function subdivision are employed as possible. Except when defining substructures for cost efficiency which eliminates replication of code, subroutines should not be utilized.

The understanding and maintenance or modification of a function is enhanced if it is structured in a step-by-step process which is easily readable. Therefore, sublevels and multidirection branches should not be utilized except where operating efficiencies are achieved. Exhibit 15 illustrates a task structure which has additional sublevels because (1) the functions are stored in a file in canonical form and (2) the task uses the APL N-task (⎕*RUN*) facility. Notice the number of levels that these conventions introduce into the task. This task was programmed using canonical representation to limit the work-sapce byte utilization and the N-task facility to reduce the APL charges being incurred (over 200 computer charge units were being incurred per exercise). With the additional levels of functions, maintenance and enhancement activities become more time consuming and difficult. When such options are chosen, the analyst and consultant should attempt to assess the trade-off value between operating savings and the incremental maintenance and enhancement costs. A simple method for determining a breakeven point is shown in Exhibit 16. The breakeven point shows the number of hours of maintenance and/or enhancement on-line time which would be required before it becomes more efficient not to implement the option. The results of the analysis should be viewed as providing a range of sensitivity for the better alternative. In some cases, the analysis will yield numbers which do not overwhelmingly support one specific direction. In this instance, the choice is left to the analyst. In any case, unless operating costs dictate sublevels, functions should contain as minimal a number of subroutines as possible.

In addition to task structures, function definitions should be provided. Function definition is quite similar to task definition in that each function requires input, performs a process, and provides output. A format for function definition is shown in Exhibit 17. This format should be completed by the consultant and reviewed by the analyst. Functions should be defined as control, process or utility functions for which a definition follows:

(1) Control: controls the execution of, and records the status of, the progress of a task.

(2) Process: an analytical algorithm or APL process (i.e., sorting) which manipulates the input data.

(3) Utility: any function which is used in more than one task to perform primarily an APL or programming step such as *DIVIDE*, *DO∆LOOP*, *INPUT*, et cetera.

This process will enable the analyst and consultant to agree that the system requirements have been interpreted correctly.

By refining the task definition into function definitions, more detailed file and workspace definitions should become available. Exhibit 18 displays the refinement of the file and workspace definitions presented in the design phase discussion. As the interaction of development continues, these definitions should be maintained as they will ultimately be the written form of documentation.

Once the task structures, function definitions, file and workspace definitions are refined and agreed upon between the analyst and the consultant, the consultant can proceed with the coding of the functions. The coding should be implemented by following the standard conventions that were agreed upon during the design phase. As discussed previously, the way one programs (or codes) a system is in itself a form of documentation. Therefore, the conventions should be followed consistently. In addition, on-line comments should be included in every function to enhance the on-line, self-documenting capabilities of the system. At the beginning of each function the following comments should be included:

```
        ∇R←A FUNCTION B
[1]⍝DS:     If not evident from function name, provide a description of the purpose of the function
            on the first line.
[2]⍝EI:     Define the explicit inputs (shape and value) A and B.
[3]⍝ER:     Define the explicit result (shape, value, and description of contents)
[4]⍝GV:     List global variable dependencies, i.e., S̲IZE, P̲ARAM
[5]⍝FD:     List file dependencies... '1234567 DATAFILE' ⎕STIE FN←2
[6]⍝SF:     List function dependencies (subordinate functions)
            .
            .
            .
[?]   ⍝     A comment should describe each major route, the use of temporary variables, and any
            complicated code within the function.
        ∇
```

Separate comments should always be included to describe steps in the process and intermediate variables (working variables) depending upon the difficulty of the code. At a minimum, the functions should be modular in an analytical sense and contain a descriptive comment for each module. The analyst should review the thoroughness of the code to assure that this level of documentation has been provided. During the testing phase, the analyst and the consultant should refer to the on-line documentation when "bugs" are encountered to see if the comments are concise and informative or if they should be improved.

Finalizing Documentation

Once the testing phase is complete, documentation will virtually be available if you followed the steps outlined previously. The remaining task is to integrate the documentation into a comprehensive reference tool. The following information should be available from the previous steps:

(1) Task Definition

(2) Task structure diagrams

(3) File definition

(4) Workspace definition

(5) Function definition

(6) On-line coded functions

(7) List or glossary of conceptual variables and variables for file storage

Except where testing requires modifications of the system effecting the available documentation, the necessary documentation should be available from the previous system phases. Additional sections should be included to fill out the necessary compliment of documentation. These sections are:

(1) **Introduction**: a discussion of what analytical purpose the system provides, who uses the system, and where the facilities (i.e., CPU, terminals, et cetera) are located.

(2) **Overview**: a comprehensive outlook at the system operation including the interaction of multiple user workspaces, files and user accounts within the system.

(3) **Conceptual variables**: mnemonics and descriptions of variables which when used, irrespective of the workspace or function or whether global or local, always contain data which conceptually represents the same entity. This list should be easy to create from the task and function definition and variables for file storage formats.

(4) **APL Coding Standards**: list the conventions which were agreed upon and used for programming and list any notable exceptions to the standards.

The final step is to arrange these sections into a comprehensive document. The suggestion that we make, based on our experience in trying to make the document not just informative, but more like a cook-book approach to resolving problems, is to organize the book so all pertinent task items are found together. Therefore, each task should be listed separately in the book with the following related items with each task:

(1) Task definition format

(2) Task structure diagrams

(3) Function definition

(4) Hardcopy of APL code for each function including a cross-reference (see documentation on I.P. Sharp's 7 *XREF* library workspace)

All functions which are subordinate to the task except utility functions should be included in the task listing. The task definition section then becomes the major section of system written documentation and can be integrated into a functional document which is outlined in Exhibit 19.

Summary

A system which is easy to maintain, modify and enhance is a result of:

- Clear and concise communication between the analyst and consultant during the construction of the system

- Recognizing that documentation is a system requirement

- Conceiving documentation as part of a project prior to any coding

- Programming documentation into the APL code (first echelon) through structure, consistency, flexibility and strategically placed comments

- Developing an integrated reference tool (second echelon) from the by-products of the system design and development phases.

Exhibit 20 summarizes the approach presented in this discussion by integrating the recommended steps for each system phase into a Systems Project Procedure chart. The suggested formats for these procedures are not rigid, as some modifications would be appropriate to adapt the guidelines to your specific application depending upon the facilities selected and the level of difficulty incorporated into the logistics of the system. The procedures are, however, important to recognize and integrate into your approach if you wish to develop a complete system which is easy to maintain and enhance. In the long run, following procedures such as these will probably reduce the number of iterations required to develop the system. This makes the system available sooner to the corporation enabling better decisions and reduces the development charges incurred by the system. For those of you who find job security in not providing adequate levels of documentation, consider that you might be the one who ultimately becomes baffled over what you did when you attempt to maintain or enhance the system. Or because of a lack of documentation you might maintain your job security in lieu of moving to a new and more challenging -- both financially and psychologically -- rewarding job!

EXHIBIT 1

THE BLACK BOX

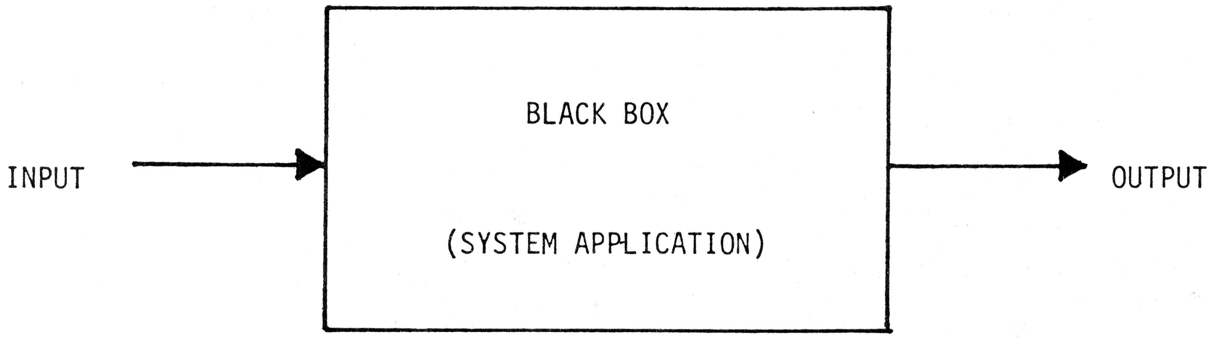
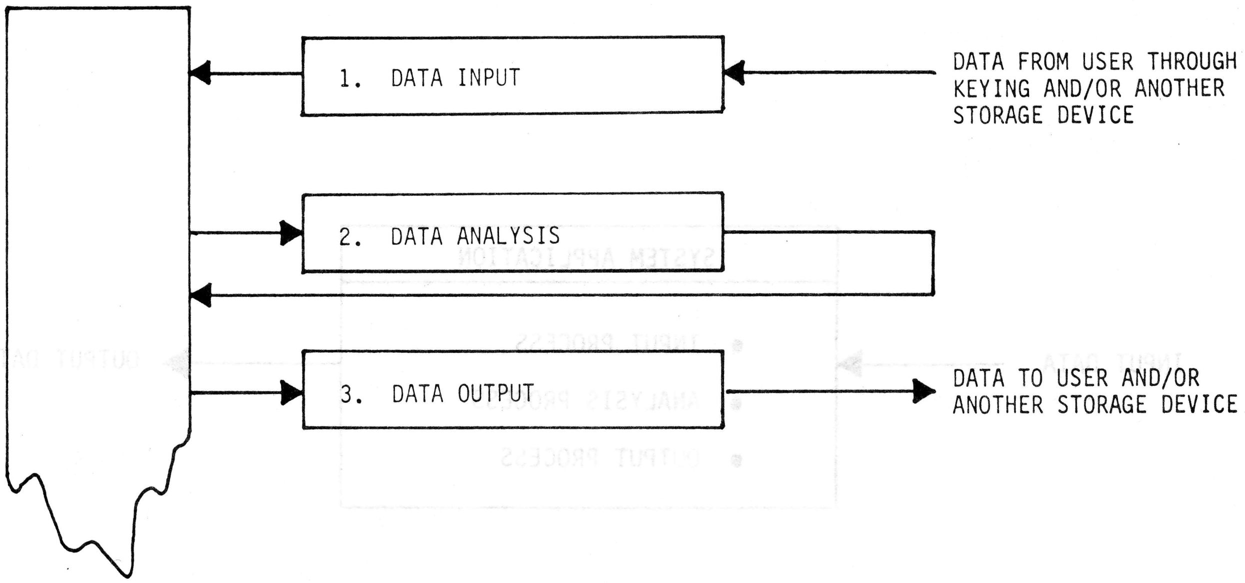

INPUT
BLACK BOX
(SYSTEM APPLICATION)
OUTPUT

EXHIBIT 2

THREE SYSTEM APPLICATIONS IN NORMAL PLANNING ENVIRONMENT


1. DATA INPUT
DATA FROM USER THROUGH KEYING AND/OR ANOTHER STORAGE DEVICE
2. DATA ANALYSIS
3. DATA OUTPUT
DATA TO USER AND/OR ANOTHER STORAGE DEVICE

DATA STORAGE DEVICE

- WORKSPACE VARIABLES
- APL FILES

APPLICATIONS INVOLVING CPU

- TERMINAL RELATED WORKSPACE PROCESSING (TTASK)
- NON-TERMINAL RELATED WORKSPACE PROCESSING
  - NTASK
  - BTASK

EXHIBIT 3

TASK DEFINITION

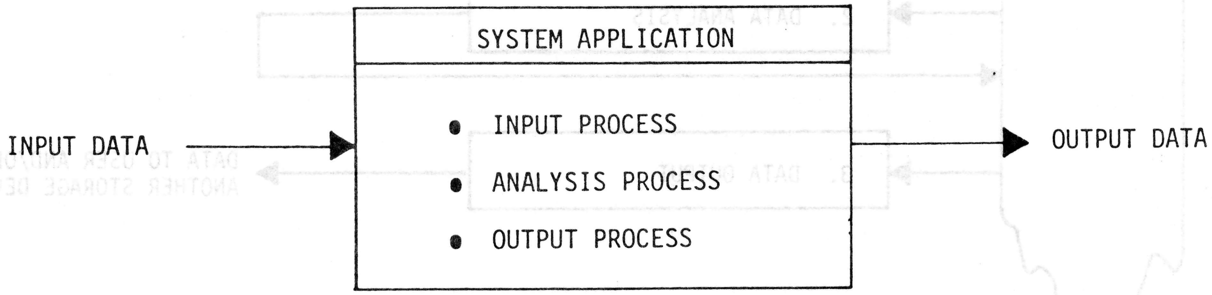

SYSTEM APPLICATION
INPUT DATA
• INPUT PROCESS
• ANALYSIS PROCESS
• OUTPUT PROCESS
OUTPUT DATA

EXHIBIT 4

EXAMPLE TASK DEFINITION

SCENARIO: A manufacturing planning group has been requested to determine labor hiring requirements for the upcoming months. From this year's experience, the company's marketing group believes that the various product sales of the company are going to continue to sell well in the marketplace. Therefore, the marketing group provides a good, better and best estimate of product sales for the next year (of course, since these are marketeers and salesmen you know the estimates are more likely to be realistic, optimistic and unbelievable!). Your information from past operational experience tells you what lead time and how much labor resource is required to manufacture one machine of each type. To provide management with the information to make a hiring decision the analyst wants to develop a system with which he can easily produce labor hiring requirements per month depending upon the various estimates. The analyst decides to have a system where an APL file retains the input data for product line populations, labor resource required to manufacture one of each of the various product types, and the lead time required to get the product in the field from time of manufacture.

EXHIBIT 4 CONT'D

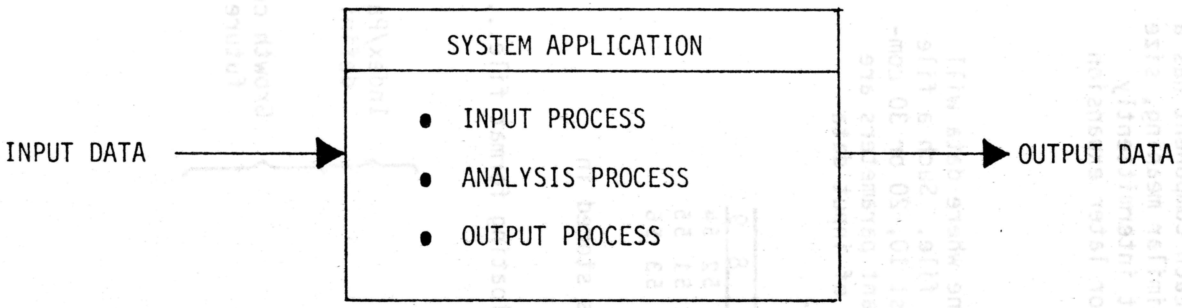
EXAMPLE TASK DEFINITION

ITEM	INPUT VARIABLES	PROCESS DEFINITION	OUTPUT DEFINITION
TASK 1 Name of Task: *LOAD*	*E* - number of exercises already defined in system. Variable is read from file. (scalar) *EX* - exercise which user selects to work with during terminal session. System defines next available exercise (*E* + 1) but allows user to override and select a previously defined exercise (scalar) Keypunched input... • *MONTHS* = number of months defined for case (*E* ρ Vector) • *PRODS* = number of different type products (*E* ρ Vector) • *POP* = number of machine sales by product line by month (*PRODS*×*MONTHS*ρMatrix) • *RESOURCE* = number of manufacturing laborers required per machine sale (*PRODS*ρ Vector) • *LEAD* = number of months after product is manufactured until it is available for sale (*PRODS*ρ Vector)	A function propmpts user to define system input through keypunching for the following items: • number of months in case • number of product lines in case • machine sales estimates by month by product line • labor resource to manufacture one machine for each product line • lead time by product line Each variable above must be designed to comprise one exercise. The program should check to verify that user inputs are in correct form and consistent with the user defined parameters (ie. if user defines 10 months in the case, then 10 machine sales estimates must be typed in for each product line. If 9 values are typed in an error retort should be returned explaining the mistake and the program should then return to the prompt for input)	Output variables to be stored in APL file: • *E* = number of exercises defined in system including exercises whose data might have been erased (scalar) • *MONTHS* = number of months defined for case (*E* ρ Vector) • *PRODS* = number of different type products (*E* ρ Vector) • *POP* = number of machine sales by product line by month (*PRODS*×*MONTHS*ρMatrix) • *RESOURCE* = number of manufacturing laborers required per machine sale (*PRODS*ρ Vector) • *LEAD* = number of months after product is manufactured until it is available for sale (*PRODS*ρ Vector)

ITEM	INPUT VARIABLES	PROCESS DEFINITION	OUTPUT DEFINITION
TASK 2 Name of Task: *CALC*	*EX* - user input variable or defined by system as next available exercise number. Represents the exercise number that the analyst will work with in current session. If the variable is system defined it is left as a global variable from TASK 1. *MONTHS* - from TASK 1 *PRODS* - from TASK 1 *POP* - from TASK 1 *RESOURCE* - from TASK 1 *LEAD* - from TASK 1	A function which processes output data from TASK 1 to determine total resources required to support product sales and print results on terminal. The calculation for a single product in a single month would be... *POP*[*P*;*M*]×*RESOURCE*[*P*;*M*] Then this would be time sequenced to account for lead times.	Output variables are formatted and printed on the terminal. User can select any combination of the following pages of data... PAGE 1: Case Parameters *EX*, *MONTHS*, *PRODS* PAGE 2&3: Input Data *POP*, *RESOURCE*, *LEAD* PAGE 4: Results • *POP*×*RESOURCE* (but data for each product line is time sequenced for lead time) • Includes total and average number of laborer-months by product line and by month • Shows hiring start month, ie. Labor resource is required 3 months before first sale for product line A. Shows by product line so hiring decisions can be made unique to manufacturing needs of specific product lines

EXHIBIT 5

REFINEMENT OF TASK DEFINITION



- Variable definition from user context (conceptual)
- Identify name to be used when coding consistent with naming conventions
- Identify and define each dimension:
  - -number of dimensions
  - -conceptual definition of each
  - -limits of each including whether limit is fixed or variable
- Identify source of variable
  - -input from user prompt
  - -workspace variable
  - -read from file

- Conceptual description of system application
- Process and/or algorithm description (ie. flow diagram)
- User conventions
  - -NTASK, TTASK, BTASK options
  - -Terminal or highspeed print options
  - -Input options
- Description of prompts
  - -Options
  - -Validation checks
  - -Unacceptable input

- Variable definition from user context (conceptual)
- Identify name to be used when coding consistent with naming conventions
- Identify and define each dimension:
  - -number of dimensions
  - -conceptual definition of each
  - -limits of each including whether limit is fixed or variable
- Identify destination of variable
  - -workspace variable
  - -stored in file
  - -print and discard
- Report formats

EXHIBIT 6

FILE FORMAT CONCEPTS

- FIXED FORMAT

A file which contains exercise, case or report data and always has a fixed number of components. Each component has a specific use and always contains data of similar meaning, size and shape. Often blank components are left intermittently throughout the file (contains 0 or ⍳0) for later expansion of the file.

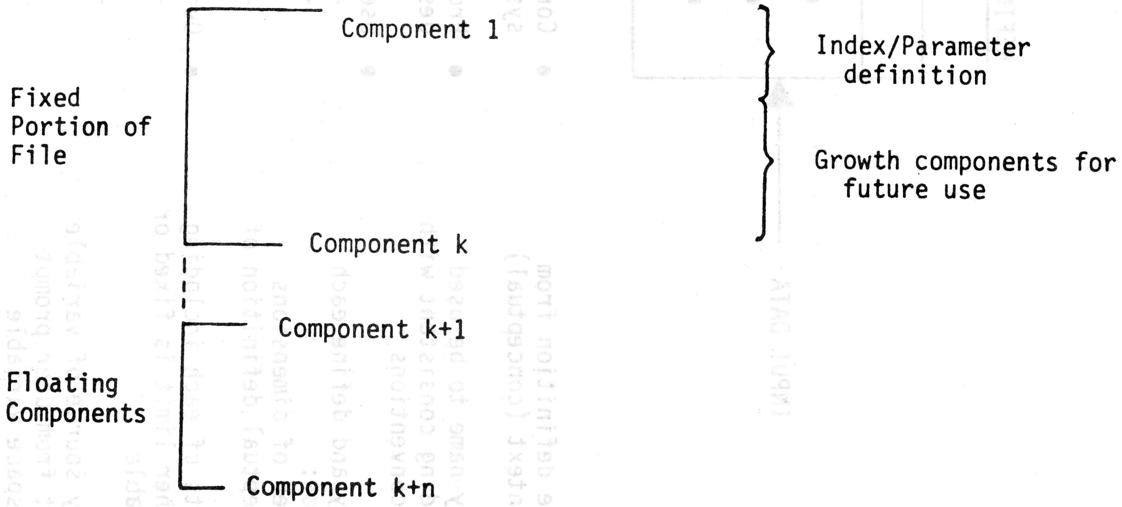
- FLOATING FORMAT

A file which uses indexing to determine where data will next be stored (⎕*APPEND* or ⎕*REPLACE*) in file. Such a file will usually have a fixed portion (ie. first 10, 20 or 30 components) where index information and constant parameters are stored. In the example case, an index map of input data storage components might look as follows...

Variables	EXERCISE								
	1	2	3	4	5	6	7	8	9
RESOURCE	30	34	35	41	38	45	49	52	54
POP	31	32	40	37	42	46	48	51	55
LEAD	33	36	39	44	43	47	50	53	56

* Variable *POP* for exercise 3 is stored in component 40

Below is a graphic representation of the floating format file...


Component 1
Index/Parameter definition
Fixed Portion of File
Growth components for future use
Component k
Component k+1
Floating Components
Component k+n

* ⎕*APPEND* to file if components k+1 to k+n are all used. ⎕*REPLACE* to file if any component number from k+1 to k+n is in empty components vector (a vector which holds the component number of those components which were used at one time but now contain data for an obsolete exercise)

EXHIBIT 7

FILE FORMAT SELECTION CRITERIA

COST		BENEFIT	
FIXED	FLOATING	FIXED	FLOATING
• Not easily enhanceable	• Not easy to use during maintenance • More complex documentation	• Easy to use during maintenance • Easy to document	• Easily enhanceable by virtue of its flexibility to add data to storage location maps

FLOATING VERSUS FIXED PREFERENCE

• Not easy to use during maintenance	• Devise routines (ie. *R←GET* 1 2 would read the component as described by the index map row 1, column 2) to ease searching for data
• More complex documentation	• Minor problem
• Easily enhanceable	• Major "cost-avoider"

EXHIBIT 8

VARIABLES FOR FILE STORAGE FROM EXAMPLE CASE

VARIABLE NAME	VARIABLE SHAPE	DESCRIPTION
		-- CONCEPTUAL VARIABLES --
E	Scalar	The number of exercises currently defined by the system including exercises whose data might have been erased.
MONTHS	Vector	Vector of length equal to *E* which contains the number of months to be considered in an exercise. Position in the vector corresponds to the exercise number. For example, *MONTHS*[3] is the number of months for which input data will be provided and output data calculated in Exercise 3.
PRODS	Vector	Vector of length equal to *E* which contains the number of product lines to be considered in an exercise. Position in the vector corresponds to the exercise number.
POP	Matrix	A matrix of two dimensions, *PRODS*×*MONTHS*, for a given exercise containing the number of machine sales for each product line expected in each forecast month.
RESOURCE	Vector	A vector equal in length to the number of product lines (*PRODS*) designated for a given exercise. Contains the number of manufacturing laborers required per machine sale for each product line in the exercise.
LEAD	Vector	A vector equal in length to the number of product lines (*PRODS*) designated for a given exercise. Contains the number of months from manufacture that elapse until a machine is available for sale for each product line.
		-- WORKING VARIABLES --
INCOMP	Matrix	Two dimensional with the number of rows being equal to the number of variables that are specific to an exercise (3 from above) and the number of exercises existing in the system. The matrix *INCOMP* contains the component number where the data for a specific variable (ie. *LEAD*) for a specific exercise (ie. exercise 4) is stored.
EMPTY	Vector	A vector containing component numbers that contain obsolete data.

EXHIBIT 9

FILE DEFINITION

NAME:
- Is file name fixed or is it variable per users' choice.
- If file name is fixed choose name.

TIE MNEMONIC AND NUMBER:
- Will a mnemonic be used.
- Will the tie number be fixed or will the tie number be variable.
- If tie number is variable, what is logic for assignment.

SECURITY RESTRICTIONS:
- Should a passnumber be placed on file.
- Should limited access be assigned to file for use by other system users, etc.

TIED BY:
- Latent expression in workspace.
- Functions in workspace.
- Which workspaces and which functions.

DESCRIPTION:
- What type data is stored in file.
- Describe file interaction among user accounts (if there are to be more than one). For example, is file to be shared among users, can each user account have its own copy, et cetera.

INITIALIZATION PROCEDURE:
- Is a function to be designated to create and initialize a new file or is it to be done manually.

LISTING:
- Component Description
  - Component Number
  - Shape
  - Description: list necessary information to understand what the contents of the component contains, where it comes from, and where and how it is used.

EXHIBIT 10

FILE DEFINITION EXAMPLE CASE

NAME: The file name should be variable depending upon user choice. Generically, it should be referred to as the DATAFILE.

TIE MNEMONIC AND NUMBER: A mnemonic should be used and a constant tie number designated.

SECURITY RESTRICTIONS: Any autotie functions should allow the user to define a passnumber so security can be placed on the file since the sales forecast data is deemed sensitive when it includes new product lines that have not yet been placed in the market. The passnumber should be stored in a global workspace variable. The autotie function should be locked so no one knows what the use is for the global variable.

TIED BY: Tied by latent expression in user workspace. The latent expression calls a tie function which automatically share ties the file name which was previously stored by the user in the global workspace variable *FNAME*.

DESCRIPTION: DATAFILE contains multiple sets of input data and exercise definition parameters needed to generate multiple hiring plan exercises. DATAFILE follows a floating format structure (components 1-29 fixed, 30 and over float) allowing multiple exercise definition. The number of exercises is limited only by the limitations of file size on user account. Multiple DATAFILES can be used on each user account because the names of such files are allowed to vary.

INITIALIZATION PROCEDURE: Initialization is to be automatically performed by a function which resides in the user workspace. The initialization function creates a file, prompts the user for name, byte allocation and a description (component 1 of file) of the file, and then initializes the fixed components of the file.

LISTING: Component description:

COMPONENT	SHAPE	DESCRIPTION
1	50ρ Character Vector	Alphanumeric file identification comments.
2	Numeric Scaler	Conceptual variable *E*
3	*E*ρ Numeric Vector	Conceptual variable *MONTHS*
4	*E*ρ Numeric Vector	Conceptual variable *PRODS*
5-9	(ι0)	Not yet defined.
10	(3,*E*)ρ Matrix	Input component map. Working variable *INCOM*
11	(Non-fixed) ρVector	Working variable empty.
12-29	(ι0)	Not yet defined.

EXHIBIT 11

WORKSPACE DEFINITION WITH EXAMPLE CASE

DEFINITION

NAME:
- Pick a name which generically represents the use of the workspace.

DESCRIPTION:
- What type of tasks are performed by workspace.

GLOBAL VARIABLES:
- File tie mnemonics
- Other variables

{ Include initial shape and value

- Are variables stored in workspace or defined by a ⎕*LX* function using ⎕*PACK* facility.

TASKS: Lists the TASKS which can be utilized in this workspace.

LATENT EXPRESSION:
- Define contents of latent expression
  - Autotie file functions.
  - Variable initialization functions.

EXAMPLE CASE

NAME: *HIPLAN* ... stands for Hiring Planning Model

DESCRIPTION: This workspace contains all the functions and variables required to perform the tasks of the Hiring Planning Model.

GLOBAL VARIABLES:
- DATAFILE tie mnemonic *FN*1←1
- DATAFILE name to be autotied *FNAME*

TASKS:
- *LOAD* ... allows user to load exercise data into a DATAFILE
- *CALC* ... allows user to execute calculation and print out exercise results.
- *INIT∆FILE* allows user to create and initiate a new DATAFILE.

EXHIBIT 12

STANDARDS FOR APL PROGRAMMING

- Variable names should always be nouns.

- Function names should always be verbs.

- The name of a conceptual variable is never used to represent more than one entity in the entire system.

- Working variables should have descriptive names if there is commonality between the use of specific working variables among functions. Otherwise, working variables should repetitively (but not within the same function) use temporary names such as *T*1, *T*10, *T*20 et cetera, to limit symbol table build-up.

- Use *I* through *N* as indeces and counters and never for anything else. Use *LMI*, *LMJ*, *LMK*, et cetera, as the limits to counters unless logical expressions or constants are more appropriate.

- Always use *R* as the result of a function unless it is appropriate to use a more descriptive name. Other uses of *R* should not be permitted to avoid the wrong values being returned by a calling function.

- Use a consistent labeling technique and never branch to ⎕*LC*, ⍳27, or an actual line number. Preferably, use label notation in sequential order and leave room between labels for expansion...

  ```
  L0:
  L10:
  L20:
  L30:
  .
  .
  .
  ```

  When a program takes two or more major routes which do not overlap (nor should they overlap), it is appropriate to label the starting statement of the route as *ROUTE*1, *ROUTE*2, *ROUTE*3: , et cetera. The route labeling can then serve as an easy reference when providing written documentation and will make the code more readable. Accordingly, the last statement in a route should be identified *END*1:, *END*2:, et cetera.

- ***END*:** should always be the only point of exit within a function.

- File tie numbers should always be in a mnemonic, *FXX*, which makes major file modification and enhancement easier.

EXHIBIT 12 CONT'D

STANDARDS FOR APL PROGRAMMING

- All conceptual and working variables should always be local if possible.

- Global variables should be underscored...*S̲IZE*. This allows the user to easily identify variables which should not be erased from workspace storage. If a local variable is assigned in a function and is global to subordinate functions it should be underscored.

- Input is always read in using ⍞(preferably a standardized function such as...*INPUT*) and validated using a standardized format if possible...

```
LO: A←⎕FI INPUT 'EXERCISE NUMBER: '
  →(0≠ρ⍞←(0≠+/A>25)/'INVALID EXERCISE...TRY AGAIN...')ρLO
```

- Flow of the program is sequential from top to bottom with modularization of the logic (by knowing input and output of a function a completely new function could be developed to replace the old one. The change should not require modifying other parts of the system unless new variables or file structure has changed).

- Standardize branching technique

  (1) Dual branch:

```
→(CODINTION TRUE)ρLABEL
```

  (2) Multiple direction branch:

```
→((COND1 TRUE),(COND2 TRUE),(COND3 TRUE))/L10,L20,L30
```

- Standardize looping techniques. Always use ≤, <, ≥,> to test to exit loop as counter may never equal limit if =is used...

```
 I←0 ◇ LMI←1↑ρABC              Initialize and define limit
LO: I←I+1                      Start loop, increment counter
 →(I≥LMI)ρLOUT                 Test for exit
 << PROCESS >>                 Process
 →LO                           Return to beginning of loop
```

- Parameter constants are never hard coded. They are assigned at the beginning of a program (or possibly read from a global variable of system constant parameters) to a variable so that the value may easily be changed. This eliminates the need to revise many lines of code because a single number changed.

- Imbedded carriage returns should not be used in any line of code except print lines.

EXHIBIT 12 CONT'D

STANDARDS FOR APL PROGRAMMING

- Multiple assignment of the same variable on a single line of code is not advised if the variable is used on the same line before it is altered or reassigned...

```
OK!           A←+/B×C∘.×D ◇ F←A×E÷S   ◇ A←H×I
           ** assign **      ** use **      ** alter **
        ...If line hit occurs A will be reassigned original value

BAD!          F←A×E÷S ◇ A←H×I ◇ A←A++/B×C∘.×D
            ** use **      ** assign **      ** alter **
        ...If line hit occurs after executing (A←H×I)
           statement F←A×E÷S would contain wrong value of A.
```

- Subroutines are discouraged unless repetitive value is inherent. However, for readability and optimum "editability" functions should be limited in length (20 to 30 lines is a good guideline).

- Avoid using complicated logistical functions (ie.⎕RUN NTASK facility) unless substantial cost savings are evident. Use of items such as the NTASK facility requires inclusion of a generally more complicated level of programming (status checks, multi-level programming, system restarts) which constrains the ability to easily enhance the analytical capabilities of a system. In addition, NTASK facilities complicate the process of tracking and recovering from errors in either the data inputs or coding. Such levels of APL facilities should be substantially justified by operating savings (see Exhibit 16).

- Function on-line documentation should include the following information...

```
          ∇R←A FUNCTION B
[1]⍝DS:   If not evident from function name, provide a description of the
          purpose of the function on the first line.
[2]⍝EI:   Define the explicit inputs (shape and value) A and B .
[3]⍝ER:   Define the explicit result (shape, value,and description of
          contents)
[4]⍝GV:   List global variable dependencies, ie. S̲IZE, P̲ARAM
[5]⍝FD:   List file dependencies... '1234567 DATAFILE' ⎕STIE FN←2
[6]⍝SF:   List function dependencies (subordinate functions)
          .
          .
          .
[?]     ⍝ A comment should describe each major route, the use of temporary
          variables, and any complicated code within the function
         ∇
```

EXHIBIT 13

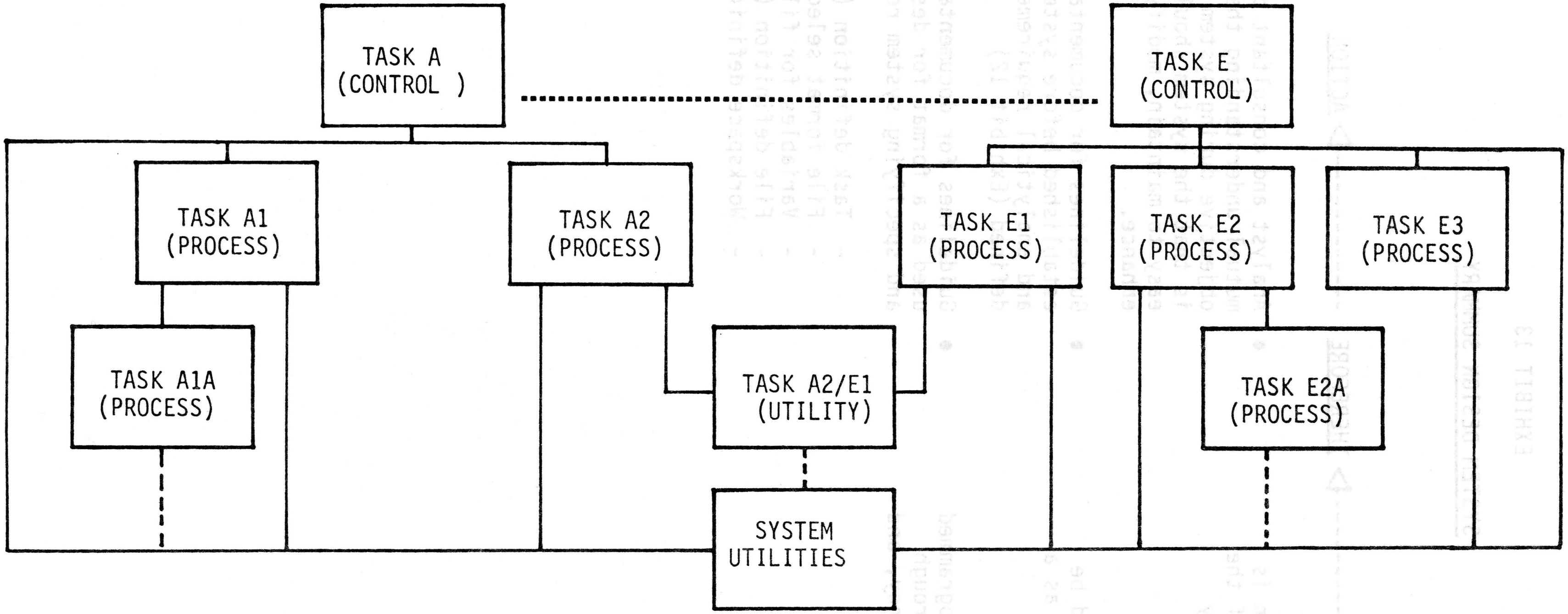
SYSTEM DESIGN SUMMARY

RULE ------------------▷ THEREFORE ------------▷ ACTION

1. Proper documentation is conceived as part of the project prior to any coding	• Analyst and consultant develop mutual understanding that one objective during system design is that the system should be easy to maintain, modify and enhance.
2. Documentation should be formally recognized as a system requirement	• Guidelines for documentation are established before system user and analytical requirements are defined (Exhibit 12)
3. Documentation is programmed into your system through structure, consistency, and flexibility.	• Guidelines for documentation are used as a format for designing and specifying system requirements - Task definition (Exhibit 3, 4, &5) - File format selection (Exhibits, 6,7) - Variables for file storage (Ex 8) - File definition (Exhibit 9&10) - Workspace definition (Exhibit 11)

EXHIBIT 14

GENERIC TASK STRUCTURE DESIGN


TASK A
(CONTROL)
TASK A1
(PROCESS)
TASK A1A
(PROCESS)
TASK A2
(PROCESS)
TASK A2/E1
(UTILITY)
SYSTEM
UTILITIES
TASK E
(CONTROL)
TASK E1
(PROCESS)
TASK E2
(PROCESS)
TASK E2A
(PROCESS)
TASK E3
(PROCESS)

FUNCTIONAL STRUCTURE

- CONTROL
- PROCESS
- UTILITY

EXHIBIT 15

EXAMPLE TASK STRUCTURE

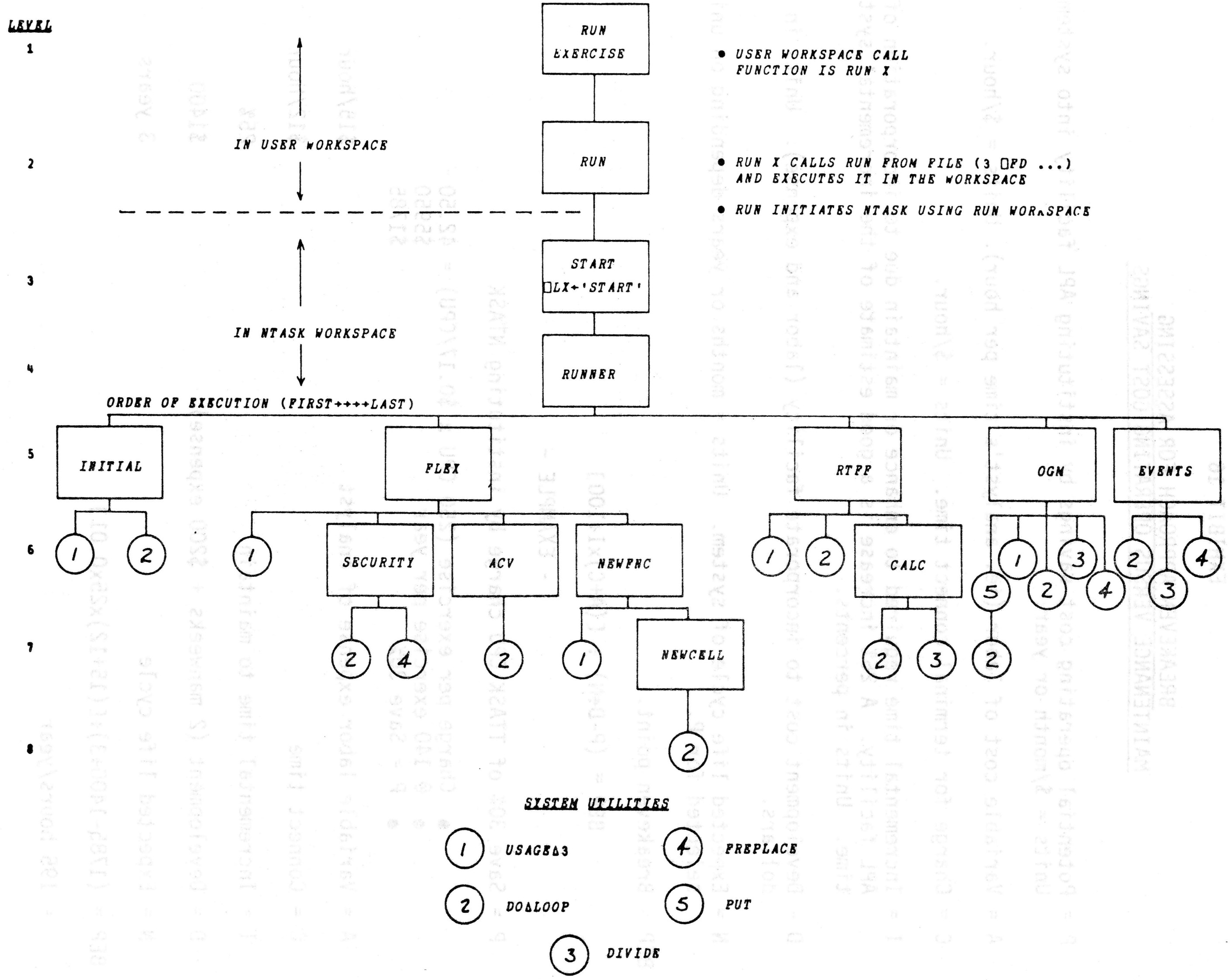

LEVEL
1
2
3
4
5
6
7
8
IN USER WORKSPACE
IN NTASK WORKSPACE
RUN EXERCISE
RUN
START
⎕LX←'START'
RUNNER
● USER WORKSPACE CALL FUNCTION IS RUN X
● RUN X CALLS RUN FROM FILE (3 ⎕FD ...) AND EXECUTES IT IN THE WORKSPACE
● RUN INITIATES NTASK USING RUN WORK∆SPACE
ORDER OF EXECUTION (FIRST→→→LAST)
INITIAL
FLEX
RTPF
OGN
EVENTS
SECURITY
ACV
NEWFNC
CALC
NEWCELL
SYSTEM UTILITIES
① USAGE∆3
② DO∆LOOP
③ DIVIDE
④ FREPLACE
⑤ PUT

EXHIBIT 16

BREAKEVEN APPROACH FOR ASSESSING MAINTENANCE VERSUS OPERATING COST SAVINGS

P = Potential operating cost savings by instituting APL facility into system. Units = $/month or year.

A = Variable cost of labor (ie. analyst's time per hour). Units = $/hour.

C = Charge for terminal connect time. Units = $/hour.

I = Incremental time required to enhance or maintain due to incorporation of APL facility. A 25% increase is a good estimate of the incremental system time. Units in percents.

D - Development cost to incorporate facility (labor and expense). Units in dollars.

N = Expected life cycle of system. Units = months or years depending on units selected for P.

BEP = Breakeven point.

$$BEP = (P - D \div N) \div [(A + C) \times I \div 100]$$

- EXAMPLE -

P = Save 30% of TTASK CPU charge by instituting NTASK

- Charge per exercise (250 CPU x $0.17/CPU) = 42.50
- @ 140 exercise per year $5950
- P = Save 30% $1785

A = Variable labor expense of analyst $15/hour

C = Connect time $12/hour

I = Incremental time to maintain 25%

D = Development (2 manweeks + $200 expense) $1400

N = Expected life cycle 3 years

$$BEP = (1785 - 1400 \div 3) \div [(15 + 12) \times 25 \times 0.01]$$

$$= 195 \text{ hours/year}$$

EXHIBIT 16 CONT'D

BREAKEVEN APPROACH FOR ASSESSING MAINTENANCE VERSUS OPERATING COST SAVINGS

- Since this calculation is based on executing 35 exercises once a quarter over the course of one month, no more than 2 hours of maintenance each week is anticipated. Since 32 hours/year (2 hrs/week x 16 we/yr = 32 hrs) is less than the BEP of 195 hours/year the APL facility should be incorporated.

EXHIBIT 17

FUNCTION DEFINITION

NAME:

TYPE:
- Control
- Process
- Utility

STORAGE LOCATION:
- Stored in Workspace Name
- Stored in component *CNX* of Name file in canoical form
- Stored in component *CNX* of Name file using ⎕*PACK*

ASSOCIATED TASK: - List tasks

PURPOSE: - Describe analytical or system purpose of function

FILE INTERACTION REQUIRED: - List all files on which function operates

EXPLICIT RESULTS: - Define shape, value and meaning of explicit results

EXPLICIT INPUTS: - Define shape, value and meaning of explicit arguments

GLOBAL VARIABLE DEPENDENCIES: - List all variables global to workspace or call function on which this function depends

SUBORDINATE FUNCTIONS CALLED: - List all functions which are called by this function

EXPLANATION: - Explain major routes of function and algorithms and difficult process/code conventions

TASK STRUCTURE DIAGRAM: - Show task diagram structure and where function fits into structure

EXHIBIT 18

REFINEMENT OF WORKSPACE AND FILE DEFINITION FORMATS

FILE DEFINITION

NAME: 1234567 *DATA* (include account number only if file is used and resides on this account for all users. If file can be on any user account omit account number).

TIE MNEMONIC AND NUMBER: *FN*←1 (if file number assignment varies explain logic for assignment).

PASSNUMBER (SECURITY RESTRICTIONS): 12345 (only document passnumber here if it is allowable for intended readers to know passnumbers.

TIED BY:
- Latent expression in workspace
- Functions in workspace
- Which workspaces and which functions?

DESCRIPTION:
- What type data is stored in file?
- Describe file interaction within user accounts of system (if there are more than one). For example, is file shared among users. Can user account have its own copy, et cetera.

INITIALIZATION PROCEDURE:
- Instructions for setting up new file.

LISTING:
- Components Overview: Consider placing an overview in an appendix with overviews for all system files if the size of the total documentation package is large. The appendix approach provides a quick reference when debugging systems. The overview should include:
  - Component number
  - Shape of data
    ie. *A*ρ Numberic Vector
    (*N*,44)ρCharacter Matrix
    (Nonfixed length) ρ Character Vector
  - Name of data stored in component
    ie. System Access Vector
    Autotied Files Name Matrix
    Product Names
- Components Description
  - Component Number
  - Shape
  - Description: list necessary information to understand what the contents of the component contains, where it comes from and where it is used.

EXHIBIT 18 CONT'D

WORKSPACE AND FILE DEFINITION FORMATS

FILE DEFINITION EXAMPLE

NAME: [ACCOUNT NUMBER]*DATAFILE*]

TIE MNEMONIC AND NUMBER: *FN*←1

PASSNUMBER: None

TIED BY: Tied by the ⎕*LX* in the user workspace *HIPLAN* (Hires Planning Model). ⎕*LX* calls the function *TIE* which automatically share ties the file name which was previously stored in the global workspace variable *FNAME*.

DESCRIPTION: *DATAFILE* contains multiple sets of input data and exercise definition parameters needed to generate multiple hiring plan exercises. *DATAFILE* follows a floating format sturcture (components 1-30 fixed, 30 and over float) allowing multiple exercise definition. The number of exercises is limited only by the limitations of the file size (bytes) on the system. Multiple *DATAFILES* can be used on each user account because the name of the file is allowed to vary. The file which will automatically be shared tied will be the one whose name is stored in the workspace global variable *FNAME*. Users can also tie and use *DATAFILES* on other user accounts given appropriate file access (⎕*RDAC*, ⎕*STAC*).

INITIALIZATION PROCEDURE: The function *INIT∆FILE*, which is located in the *HIPLAN* user workspace, will ⎕*CREATE* new *DATAFILES* and prompt the user for the appropriate input to initiate new *DATAFILES*. The initialization procedure executed by *INIT∆FILE*, ⎕*APPENDS* components 1-30 and stores the following information in each.

LISTING: Components description.....

COMPONENT	SHAPE	DESCRIPTION
1	50 Character Vector	File identification. Alphanumeric description of file in relation to analytical exercise.
2	Numeric Scalar	Exercise limit. (*E*): number of exercises defined in system including exercises whose data might have been erased. (1 + number of exercises already defined).
3	*E*ρ Numeric Vector	Period limit (*MONTHS*): number of months defined for each exercise.

EXHIBIT 18 CONT'D

WORKSPACE AND FILE DEFINITION FORMATS

LISTING: Components description... (cont'd)

COMPONENT	SHAPE	DESCRIPTION
4	*E*ρ Numeric Vector	Product limit: (*PRODS*): number of product lines defined for each exercise
5-9	(ι0)	Not yet defined
10	(3,*E*)ρ Matrix	Input component map (*INCOMP*): component numbers where the data for a specific variable is stored. ROW INPUT DATA 1 *POP* 2 *RESOURCE* 3 *LEAD* The columns correspond to the exercise number.
11	(Non-fixed)ρ Vector	Empty components (*EMPTY*): Contains floating (30 and over) component numbers that contain absolete data.
12-29	(ι0)	Not yet defined

* NOTE: The components overview would be a simple listing that could be included as an easy reference appendix displaying component number, shape, and the data name (ie. Exercise limit) as listed above under DESCRIPTION.

EXHIBIT 18 CONT'D

WORKSPACE AND FILE DEFINITION FORMATS

WORKSPACE DEFINITION

NAME: List name

DESCRIPTION: What type of tasks are performed by workspace.

GLOBAL VARIABLES: List variables that reside as global variable in workspace or are defined as global variables by ⎕*LX*. Include the shape of variable, initial contents (explain default contents if defined through logic or mathematical statement), and a conceptual description of variable.

TASKS: List the tasks which can be utilized in this workspace.

LATENT EXPRESSION:
- Define contents of latent expression
  - Autotie funcitons
  - Variable initialization functions
  - Mode definitions (ie. ⎕*OUT*)

FUNCTIONS:
- List and define functions used in workspace.
  - Which task is it used by
  - Control, process or utility

WORKSAPCE DEFINITION EXAMPLE

NAME: *HIPLAN*... for Hiring Planning Model

DESCRIPTION: This workspace contains all functions necessary to perform the available tasks of the Hiring Planning Model.

GLOBAL VARIABLES:

NAME	SHAPE	INITIAL VALUE	RESIDES IN:	DESCRIPTION
*FN*1	Scalar	1	*HIPLAN*	DATAFILE tie mnemonic
FNAME	22ρ Character Matrix		*HIPLAN*	DATAFILE name to be autotied. If *FNAME* is not defined (checks 1⎕*WS* 2 to see if variable is defined) *FNAME* autotie function asks user to input name.

EXHIBIT 18 CONT'D

WORKSPACE AND FILE DEFINITION FORMATS

NAME	SHAPE	INITIAL VALUE	RESIDES IN:	DESCRIPTION
E	Scalar	-	*DATAFILE* *CMP* 2	Number of exercises currently defined in system.

TASKS:
- *LOAD* ... allows user to load exercise data into a DATAFILE
- *CALC* ... allows user to execute calculation and print out exercise results
- *INITΔFILE* . allows user to create and initiate a new DATAFILE.

LATENT EXPRESSION: The latent expression calls two function...*TIE*, and *INITIAL*. The functions tie the DATAFILE and initialize global variables, respectively.

FUNCTIONS:

NAME	TYPE	RELATED TASK
LOAD	Control and Process	LOAD
CALC	Control and Process	CALC
INITΔFILE	Control and Process	INITΔFILE
INITIAL	Process	-
TIE	Process	-
DIVIDE	Utility	-
DOΔLOOP	Utility	-
INPUT	Utility	-
OUTPUT	Process	CALC

EXHIBIT 19

FUNCTIONAL APL TECHNICAL DOCUMENTATION FORMAT

Introduction

- What analytical purpose does system fullfill
- Who uses system
- What and where are facilities for system operation located (CPU, terminals, highspedd printers, et cetera)

Overview

- Comprehensive outlook at system operation
  - Account interaction
  - Workspace interaction
  - File interaction

Task Description - Provide the following for each task: (List alphabetically by task)

- Task Definition (Exhibit 5)
- Task structure diagram (Exhibits 14 and 15)
- For each function - excluding utilities, listing functions in either alphabetic order or in order by which they are processed by task:
  - Function definition (Exhibit 17)
  - Hard-copy of APL code with a cross-reference (*EXREF*)

File Definition - (Exhibit 18)

Workspace Definition - (Exhibit 18)

Utility Functions - Listed alphabetically

- Function definition (Exhibit 17)
- Hard-copy of APL code with a cross-reference (*EXREF*)

Variable Glossary - (Same format as presented in Exhibit 8)

- A list of the conceptual variables and universal working variables (working variables which are common among all workspaces in system)
  - Conceptual description
  - Shape and description of dimensions

Appendices

- File component overviews

EXHIBIT 20

APL SYSTEMS PROJECT PROCEDURES CHART

Design Phase: System Requirements

- Analyst and consultant develop mutual understanding that one objective during system design is that the system should be easy to maintain, modify and enhance.
- Guidelines for documentation are established before system user and analytical requirements are defined (Exhibit 12)
- Guidelines for documentation are used as a format for designing and specifying system requirements
    - Task definition (Exhibit 3, 4, &5)
    - File format selection (Exhibits, 6,7)
    - Variables for file storage (Ex 8)
    - File definition (Exhibit 9&10)
    - Workspace definition (Exhibit 11)

Development and Coding Phases: System Specifications/Coding

- Refine Task Definitions
    - Task Structure Diagram (Exhibit 14 and 15)
    - Function Definition (Exhibit 17)
      Control, process and utility functions
- Refine File Definition (Exhibit 13)
- Refine Workspace Definition (Exhibit 18)
- Review coding comments and update task, function, file and workspace definitions as required.

Finalizing Documentation: Technical Documentation Manual

- Integrate definition formats into technical docmentation reference tool *(Exhibit 19)

* A large portion of this documentation is also applicable for user documentation.

APPLICATION OF POLYNOMIAL DISTRIBUTED LAG TO "SOCIAL AND PERSONAL COSTS OF UNEMPLOYMENT"

Neil C. MacLeod
Long Range Planning
Department of National Health and Welfare
Ottawa, Ontario

Background and Introduction

It has often been hypothesized that the state of the economy affects the level of stress in a society. To the extent that unemployment and inflation bring about real hardship, increases in these rates can be expected to bring about increases in stress levels among members of that society. Therefore to the extent that increased levels of stress manifest themselves in an increase in the incidence of pathological behaviour, we might expect to find a significant relationship between fluctuations in the values of certain economic indicators and the incidence of certain social pathologies.

The analysis of such a relationship can take several forms. First, on a micro-analytic cross-sectional basis, direct cause and effect patterns can be postulated to operate within an individual. Second, an ecological level model can be hypothesized wherein the possibility of interpersonal links between, say, a state of unemployment and the committing of some crime, can be better handled. Third, a macro-level time-series model can be used wherein aggregate levels of economic and social indicators can be compared over time.

As part of the initial phase of a project at the Department of National Health and Welfare on the Social and Personal Costs of Unemployment, the third type of model described above was adopted. The initial specification of the model followed that used by Dr. M. Harvey Brenner of Johns Hopkins University in a document prepared for the Joint Economic Committee of the U.S. Congress. The analytical features of this model include the use of the polynomial distributed lag structure (or Almon lag structure) in the statistical estimation procedure, a feature which allows the effects that past and present levels of independent variables have on present levels of dependent variables to be estimated.

To carry out the analysis, a computer system was required that could handle data sets of moderate size, and, at the same time, contain a flexible, reliable, and efficient statistical package to deal with the polynomial distributed lag structure. It was decided to use the I.P. Sharp APL system for the purpose. This paper represents a summary of the uses of the Sharp system with this project, with special emphasis on the empirical findings from the PDL package.

This paper is presented in four parts. In the first, a brief outline of the method of polynomial distributed lags, including a description of the corresponding APL package, is given. In the second, some of the results of the study are presented emphasizing apparent anomalies of the PDL structure. The third section includes a more general commentary on the method, a suggested procedure for evaluating the appropriateness of the PDL structure, and the fourth a conclusion.

The opinions expressed in this paper are those of the author and in no way represent official policy positions of the Department of National Health and Welfare.

A Brief Outline of the Method

The following is a brief outline of the polynomial distributed lag method. Few technical details are included; for a more in-depth mathematical development see Foot and North (1) or Johnston (2).

The polynomial distributed lag structure is used in estimating relationships wherein the effects of independent variables are hypothesized to stretch back in time. Considering, for the sake of simplicity, the case of only one independent variable, the ordinary least squares model is:

$$\underline{Y} = X\underline{\beta} + \underline{u} \;.\;.\;.\;. (1)$$

where $\underline{Y}$ is the vector of length n of observed values of the independent variable, X is the (nx2) dimensional matrix of the independent variable, (the entries in the first column are all 1's). $\underline{\beta} = (\alpha\ \beta)$ is the 2-element vector of parameters to be estimated and $\underline{u}$ is an n-element column vector representing the error term. In this case the value of Y in year t is assumed to depend on the values of X, only in year t, i.e.:

$$Y_t = \alpha+\beta X_t + u_t, \ldots . (2)$$

If it is assumed, however, that Y_t depends not only on X_t but also on $X_{t-1}, X_{t-2}, \ldots, X_{t-T}$, then we would have

$$Y_t = \alpha+\beta_t X_t + \beta_{t-1} X_{t-1} + \ldots + \beta_{t-T} X_{t-T} + u_t, \ldots . (3)$$

In such a case the (T+2) parameters $\alpha, \beta_t, \beta_{t-1}, \ldots, \beta_{t-T}$ can be estimated in the same way as are ordinary least squares estimates with (T+1) independent variables.

If the number of observations, n, is sufficiently large with respect to the number of years lagged, T, then equation (2) should in fact be treated as an ordinary least squares problem and no special lag structure is needed. However, when such is not the case (and of course when several variables are to be lagged the size of "T" is effectively the sum of the lag lengths of the independent variables) it is worthwhile attempting to reduce the number of coefficients by imposing a structure on them.

One such structure is a polynomial function. For the sake of simplicity we will consider only the case where the polynomial is of a degree 2 (again, the more general case can be found in Foot and North (1) and Johnston (2)). In this case we force the β's to be quadratic functions of a new set of three parameters. In fact, we define:

$$\begin{aligned} \beta_t &= a_o \\ \beta_{t-1} &= a_o + a_1 + a_2 \\ \beta_{t-2} &= a_o + 2a_1 + 4a_2 \qquad \ldots (4) \\ \beta_{t-T} &= a_o + Ta_1 + T^2 a_2 \end{aligned}$$

so that we have reduced the number of parameters from (T+2) to 4-viz. α, a_o, a_1, and a_2.

Having done this, we can substitute the new "a's" into equation (3), rearrange combinations of the X variable so as to isolate the a's and then apply ordinary least squares to arrive at estimates of them. Subsequently the estimates of β's can be calculated using (4). Formulae for standard errors are found in Foot and North (1).

The PDL Program

The program which carries out the estimation procedure outlined above is stored in the `32` *PDLAG* workspace. It is, as is typical of programs in the library, a highly conversational program that is very easy to understand. The user is asked to enter the matrix of dependent and independent variables, the order of the polynomial to be imposed on the coefficients (above we used a quadratic, so the order was two), the beginning of the lag period (we used zero, but this was not necessary so, for example, the value of a dependent variable in year t could be hypothesized to depend on the values of an independent variable in years (t-2) back to (t-8)), the end of the lag period (in the outline it was T) and the choice of including constraints that the most recent or least recent of the T lag years has zero effect on the dependent variable.

An example from our work of how the program operates is shown below.

```
ENTER DEPENDENT VARIABLE
⎕:
      T1 [;2]
ENTER MATRIX OF INDEPENDENT VARIABLES WHICH ARE NOT TO BE LAGGED (0 IF NONE)
⎕:
      0
ENTER HYPOTHESIZED LAG DISTRIBUTION AND INDEPENDENT VARIABLE
⎕:
      T LAG NOW1
ANOTHER INDEPENDENT VARIABLE TO BE LAGGED?
Y
ENTER HYPOTHESIZED LAG DISTRIBUTION AND INDEPENDENT VARIABLE
⎕:
      T LAG NOW2
ANOTHER INDEPENDENT VARIABLE TO BE LAGGED?
Y
ENTER HYPOTHESIZED LAG DISTRIBUTION AND INDEPENDENT VARIABLE
⎕:
      T LAG NOW3
ANOTHER INDEPENDENT VARIABLE TO BE LAGGED?

ALIGN PAPER
```

```
ORDINARY LEAST SQUARES

                 ESTIMATED          STANDARD
                COEFFICIENT          ERROR           T-VALUE

CONSTANT         -0.12426           1.21976         -0.10187
   A 1            0.49924           0.10510          4.75012
   A 2           -0.36491           0.14549         ¯2.50804
   A 3            0.05531           0.02886          1.91635
   A 4            0.05179           0.05209          0.99417
   A 5           ¯0.00266           0.05263         ¯0.05047
   A 6           ¯0.00439           0.00975         ¯0.50191
   A 7            0.00585           0.00246          2.37542
   A 8           ¯0.00487           0.00369         ¯1.32187
   A 9            0.00074           0.00074          0.98746

R-SQUARED.....................                      0.89052
RBAR-SQUARED..................                      0.86154
F-STATISTIC ( 9, 34)..........                     30.72844
STANDARD ERROR OF THE ESTIMATE                      1.07890
COEFFICIENT OF VARIATION......                     11.41579
DURBIN-WATSON STATISTIC.......                      1.41305

DISTRIBUTED LAG INTERPRETATION : LAGGED VARIABLE 1

LAG      COEFFICIENT        STD. ERROR          T-VALUE
 0           0.49924           0.04683         10.66032
 1           0.18964           0.01387         13.66910
 2          ¯0.00933           0.03217         ¯0.29012
 3          ¯0.09769           0.03240         ¯3.01527
 4          ¯0.07543           0.01438         ¯5.24657
 5           0.05745           0.04601          1.24867
```

```
SUM OF LAG COEFFICIENTS 0.5638840774
STANDARD ERROR OF SUM     0.02336359624

DISTRIBUTED LAG INTERPRETATION : LAGGED VARIABLE 2

 LAG     COEFFICIENT        STD. ERROR          T-VALUE
  0          0.05179          0.03211           1.61306
  1          0.04424          0.01506           2.93806
  2          0.02691          0.02061           1.30536
  3         ¯0.00020          0.02063          ¯0.00992
  4         ¯0.03710          0.01611          ¯2.30334
  5         ¯0.08378          0.03447          ¯2.43074

SUM OF LAG COEFFICIENTS 0.001850677559
STANDARD ERROR OF SUM     0.04807647543

DISTRIBUTED LAG INTERPRETATION : LAGGED VARIABLE 3

 LAG     COEFFICIENT        STD. ERROR          T-VALUE
  0          0.00585          0.00120           4.88468
  1          0.00172          0.00039           4.39025
  2         ¯0.00095          0.00097          ¯0.98260
  3         ¯0.00215          0.00094          ¯2.28228
  4         ¯0.00187          0.00036          ¯5.18108
  5         ¯0.00013          0.00136          ¯0.09229

SUM OF LAG COEFFICIENTS 0.002473213968
STANDARD ERROR OF SUM   0.0001447223333
```

A Summary of the Results

In this section a description of some of the results is given. As a background, graphs of the independent variables for Canada and the dependent variables by region and by age are also provided. The graphs of the independent variables are shown below while the other graphs are included as required throughout the description.

In the first part the effects of inflation, unemployment, and per capita income on (separately) the mortality rate, the suicide rate, the homicide rate, the rate of first admissions to mental hospitals, and the rate of mortality due to cirrhosis of the liver are described within the five geographic **regions** of Canada. In the second part the results of the same kind of analysis for mortality, suicide and homicide rates, and the rate of convictions for indictable offences within **age** groups is shown. And in the final part a chart summarizing the direction of effects that each independent variable has is provided.

FIGURE 1

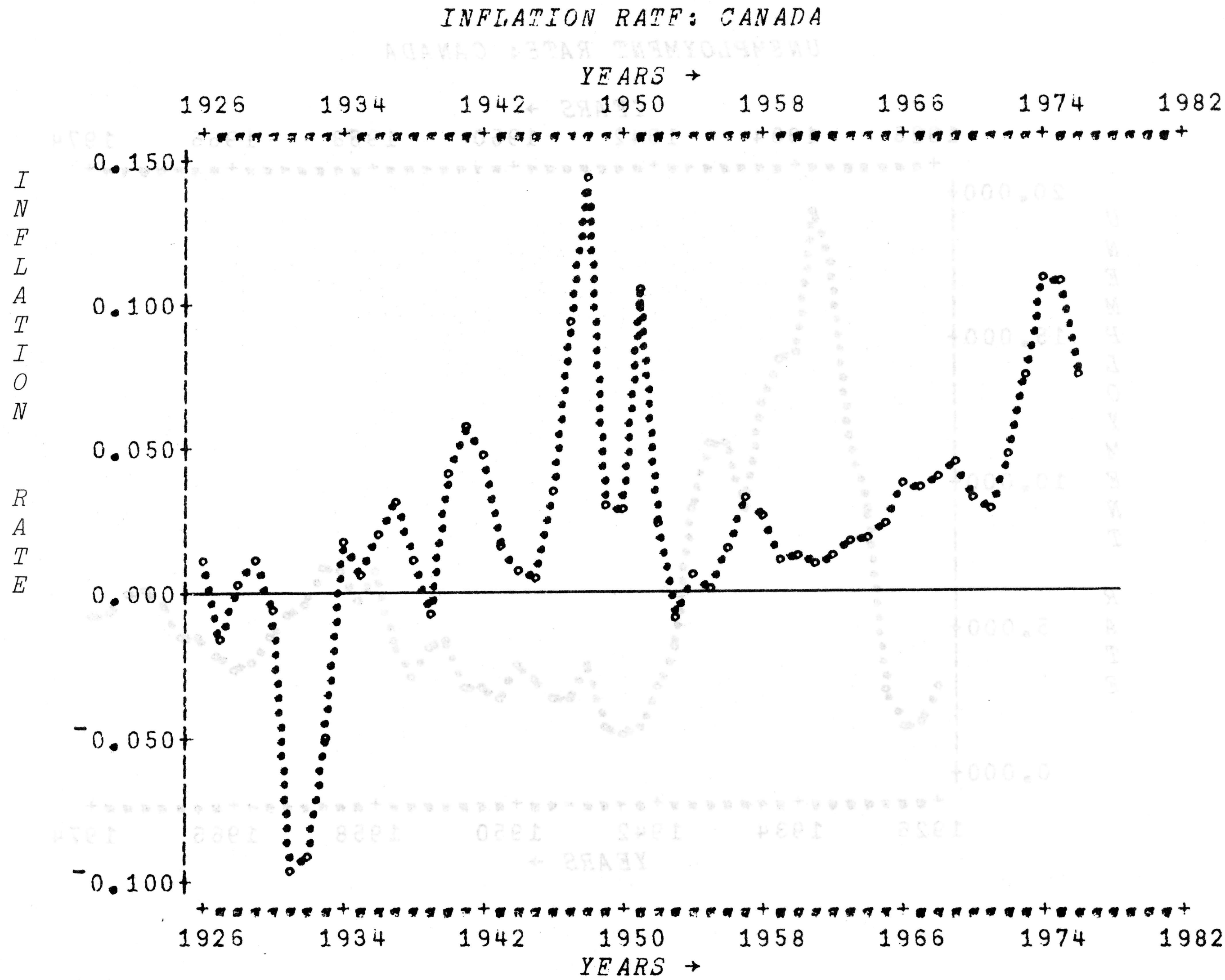

INFLATION RATE: CANADA
YEARS →
1926
1934
1942
1950
1958
1966
1974
1982
INFLATION RATE
0.150
0.100
0.050
0.000
-0.050
-0.100
1926
1934
1942
1950
1958
1966
1974
1982
YEARS →

FIGURE 2

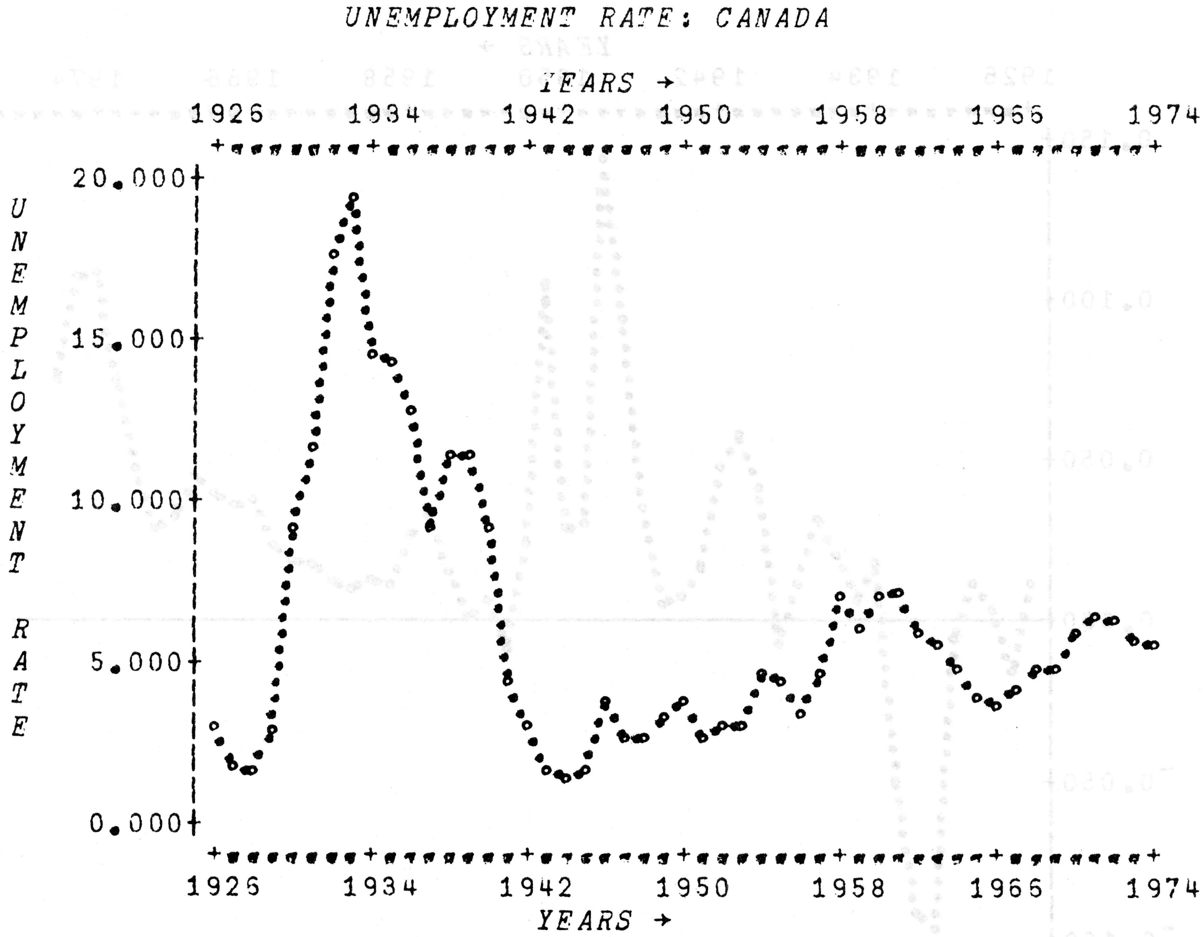
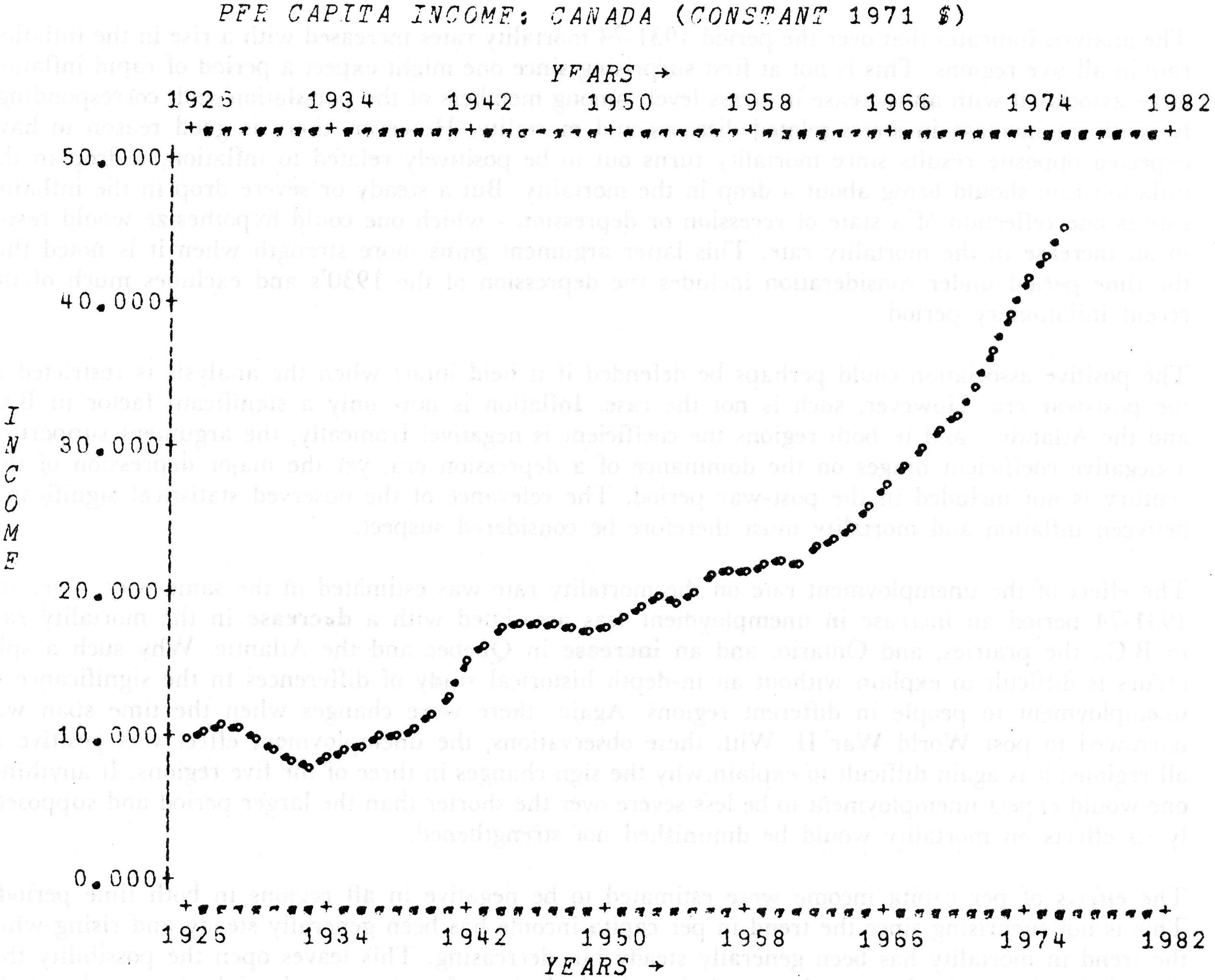

UNEMPLOYMENT RATE: CANADA
YEARS →
1926
1934
1942
1950
1958
1966
1974
UNEMPLOYMENT RATE
20.000
15.000
10.000
5.000
0.000
1926
1934
1942
1950
1958
1966
1974
YEARS →

FIGURE 3


PER CAPITA INCOME: CANADA (CONSTANT 1971 $)
YEARS →
1926
1934
1942
1950
1958
1966
1974
1982
50.000
40.000
30.000
20.000
10.000
0.000
INCOME
YEARS →

Results by Region

1. Mortality

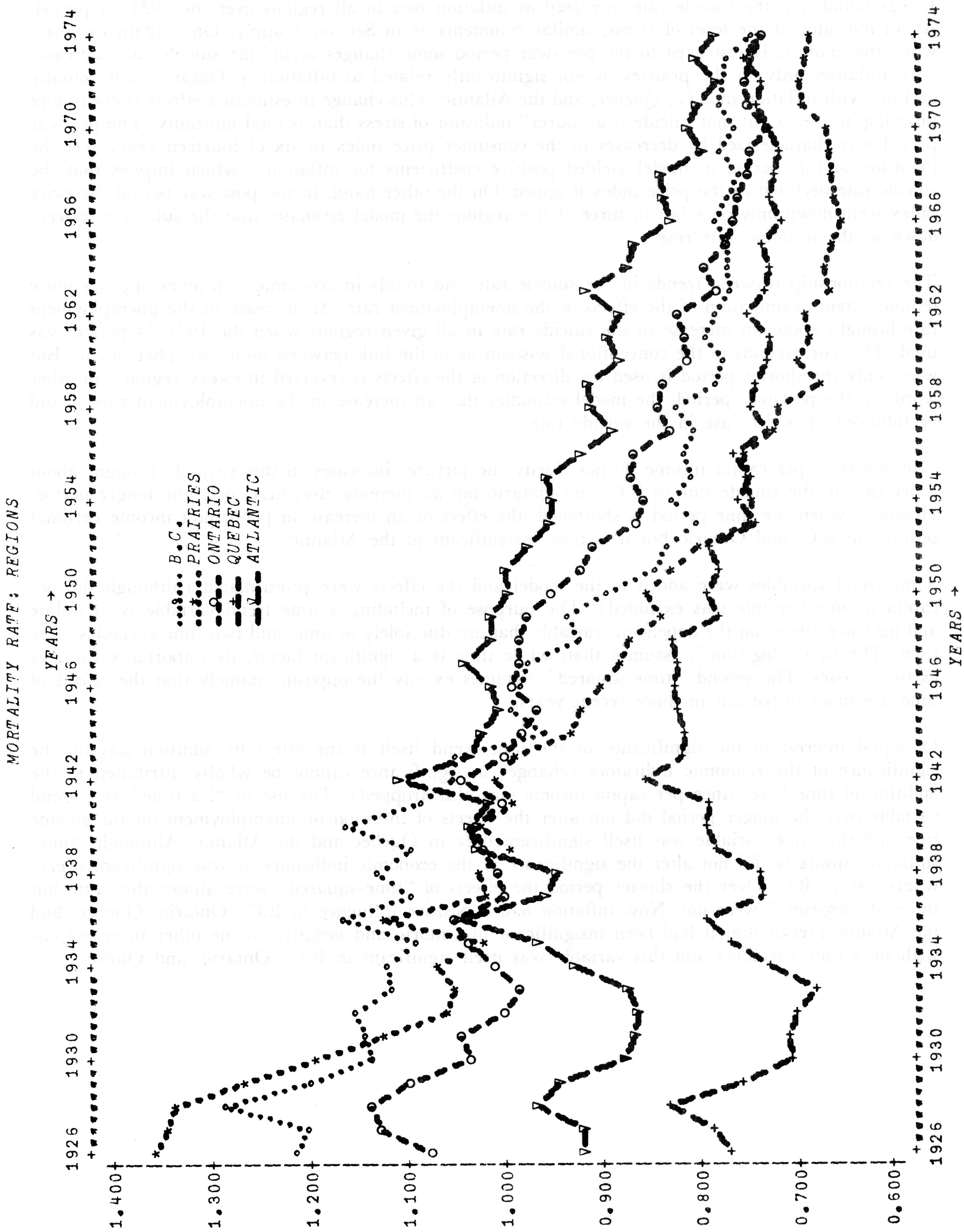
The analysis indicates that over the period 1931-74 mortality rates increased with a rise in the inflation rate in all five regions. This is not at first surprising since one might expect a period of rapid inflation to be associated with an increase in stress levels among members of the population and, correspondingly, with an increase in stress-related diseases and mortality. However, there is good reason to have expected opposite results since mortality turns out to be positively related to inflation, a drop in the inflation rate should bring about a drop in the mortality. But a steady or severe drop in the inflation rate is one reflection of a state of recession or depression - which one could hypothesize would result in an increase in the mortality rate. This latter argument gains more strength when it is noted that the time period under consideration includes the depression of the 1930's and excludes much of the recent inflationary period.

The positive association could perhaps be defended if it held intact when the analysis is restricted to the post-war era. However, such is not the case. Inflation is now only a significant factor in B.C. and the Atlantic - and in both regions the coefficient is negative! Ironically, the argument supporting a negative coefficient hinges on the dominance of a depression era, yet the major depression of this century is not included in the post-war period. The relevance of the observed statistical significance between inflation and mortality must therefore be considered suspect.

The effect of the unemployment rate on the mortality rate was estimated in the same way. Over the 1931-74 period an increase in unemployment was associated with a **decrease** in the mortality rate in B.C., the prairies, and Ontario, and an **increase** in Quebec and the Atlantic. Why such a split occurs is difficult to explain without an in-depth historical study of differences in the significance of unemployment to people in different regions. Again, there were changes when the time span was narrowed to post World War II. With these observations, the unemployment effect was positive in all regions; it is again difficult to explain why the sign changes in three of the five regions. If anything, one would expect unemployment to be less severe over the shorter than the larger period and supposedly its effects on mortality would be diminished not strengthened.

The effects of per capita income were estimated to be negative in all regions in both time periods. This is not surprising since the trend in per capita income has been generally steady and rising while the trend in mortality has been generally steady but decreasing. This leaves open the possibility that the relationship is in fact spurious and that the inclusion of a time trend might have reduced the significance of this effect.

FIGURE 4



MORTALITY RATE: REGIONS
YEARS →
B.C.
PRAIRIES
ONTARIO
QUEBEC
ATLANTIC
1926
1930
1934
1938
1942
1946
1950
1954
1958
1962
1966
1970
1974
1.400
1.300
1.200
1.100
1.000
0.900
0.800
0.700
0.600
YEARS →

2. Suicide

It was found that the suicide rate increased as inflation rose in all regions over the 1931-74 period. As an indicator of the level of stress, similar comments as in Section 1 apply. Once again however, when the analysis is restricted to the post-war period some changes occur: the suicide rate increases with inflation **only** in the prairies, is not significantly related to inflation in Ontario, and actually declines with inflation in B.C., Quebec, and the Atlantic. This change in estimated effects is even more puzzling, to the extent that suicide is a "purer" indicator of stress than is total mortality. The pre-war period was characterized by decreases in the consumer price index in six of fourteen years. Yet the inclusion of this period in model yielded positive coefficients for inflation - which implies that the suicide rate declined as the price index dropped. On the other hand, in the post-war period the price index went down only once but in three of five regions the model estimates that the suicide rate went down as the inflation rate rose.

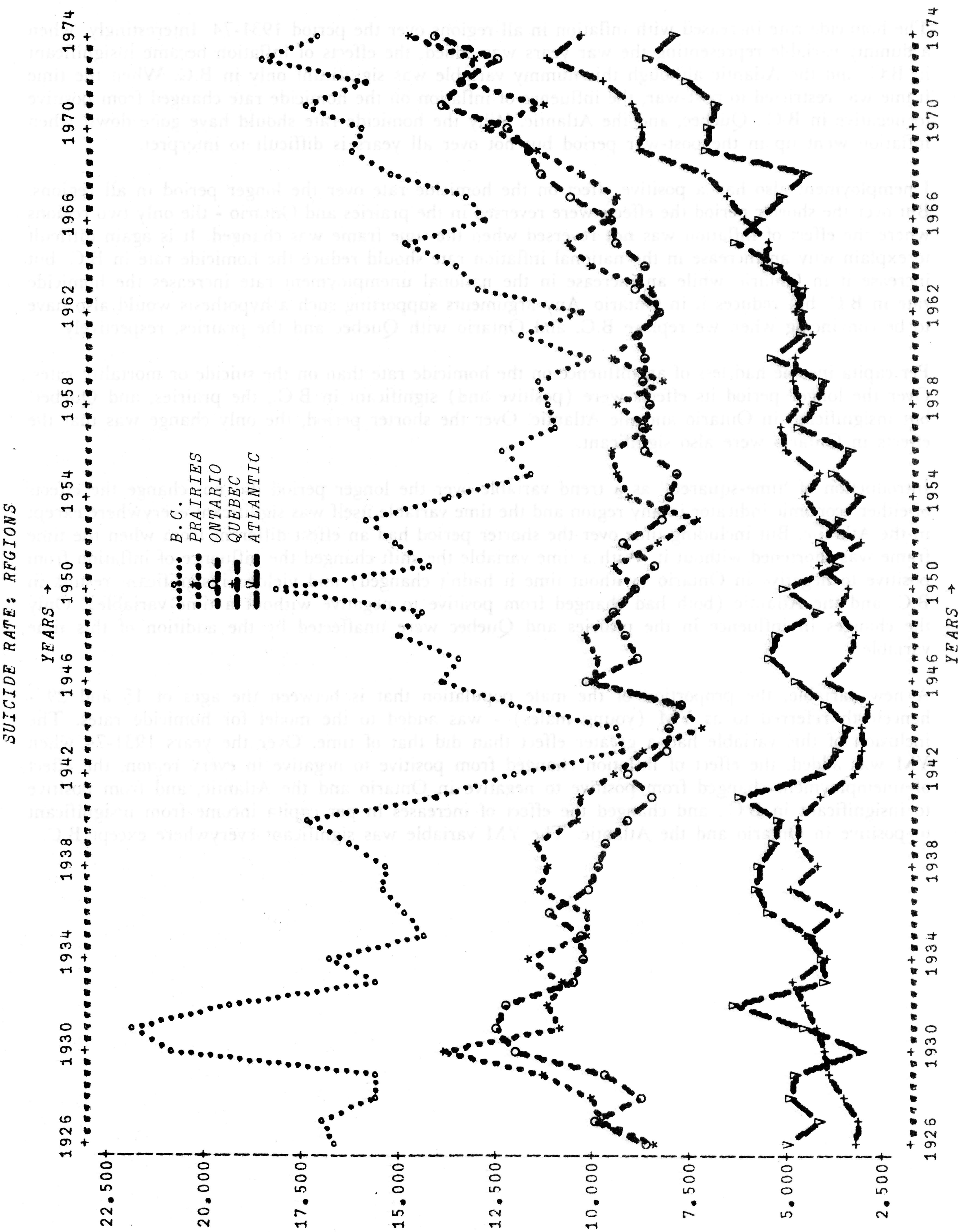
The relationship between trends in the suicide rate and trends in economic indicators appears more tenuous upon examination of the effects of the unemployment rate. An increase in the unemployment rate brought about an increase in the suicide rate in all given regions when the 1931-74 period was used. This corresponds to the conventional wisdom as to the link between these two phenomena. But when only the shorter period is used the direction of the effects is reversed in **every** region - in other words in the post-war period, the model estimates that an increase in the unemployment rate would be followed by a decrease in the suicide rate.

The effects of per capita income do not clarify the picture. Increases in this variable brought about a decrease in the suicide rate in B.C. and Ontario but an increase elsewhere, over the longer period. However, when the time period is shortened, the effect of an increase in per capita income becomes positive in B.C. and Ontario but it is now insignificant in the Atlantic.

Time trend variables were added to the model and the effects were re-estimated (although the per capita income variable was excluded). The purpose of including a time trend variable is to isolate and measure effects on the dependent variable that are due solely to time, and two time variables were used. The first, "log-time", assumes that, while time is a significant factor, its importance declines as time passes. The second, "time-squared", assumes exactly the opposite, namely that the effects of time are more important in more recent years.

Of equal interest to the significance of the time trend itself is the effect its addition has on the significance of the economic indicators (changes in significance cannot be wholly attributed to the addition of time here, since per capita income was also dropped). The use of "log-time" as a trend variable over the longer period did not alter the effects of inflation or unemployment on the suicide rate and this time variable was itself significant only in Quebec and the Atlantic. Although "time-squared" similarly did not alter the significance of the economic indicators, it was significant everywhere except B.C. Over the shorter period the effects of "time-squared" were almost the same but those of "log-time" were not. Now inflation had a positive influence in B.C., Ontario, Quebec, and the Atlantic (recall that it had been insignificant in Ontario and negative in the other three regions without a time variable) and this variable was itself significant in B.C., Ontario, and Quebec.

FIGURE 5


SUICIDE RATE: RFGIONS
YEARS →
B.C.
PRAIRIES
ONTARIO
QUEBEC
ATLANTIC
22.500
20.000
17.500
15.000
12.500
10.000
7.500
5.000
2.500
1926
1930
1934
1938
1942
1946
1950
1954
1958
1962
1966
1970
1974
YEARS →

3. Homicide

The homicide rate increased with inflation in all regions over the period 1931-74. Interestingly, when a dummy variable representing the war years was added, the effects of inflation became insignificant in B.C. and the Atlantic although this dummy variable was significant only in B.C. When the time frame was restricted to post-war, the influence of inflation on the homicide rate changed from positive to negative in B.C., Quebec, and the Atlantic. Why the homicide rate should have gone down when inflation went up in the post-war period but not over all years is difficult to interpret.

Unemployment also had a positive effect on the homicide rate over the longer period in all regions. But over the shorter period the effects were reversed in the prairies and Ontario - the only two regions where the effect of inflation was **not** reversed when the time frame was changed. It is again difficult to explain why an increase in the national inflation rate should reduce the homicide rate in B.C. but increase it in Ontario while an increase in the national unemployment rate increases the homicide rate in B.C. but reduces it in Ontario. Any arguments supporting such a hypothesis would also have to be convincing when we replace B.C. and Ontario with Quebec and the prairies, respectively.

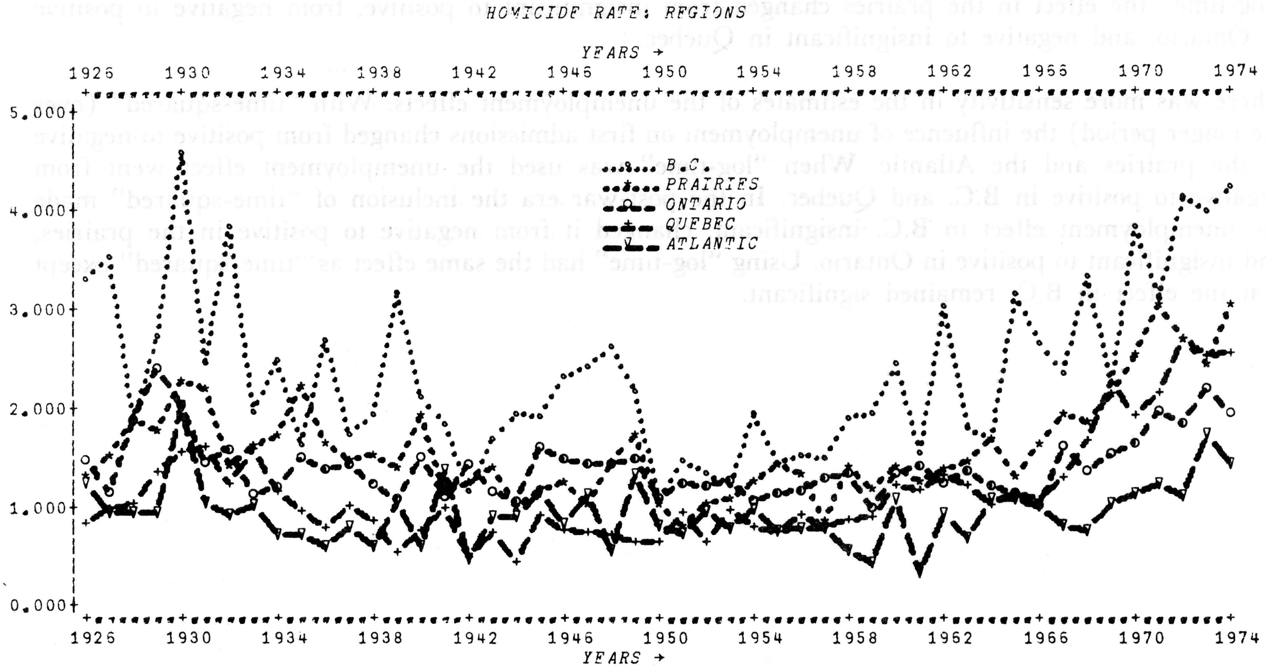
Per capita income had less of an influence on the homicide rate than on the suicide or mortality rates. Over the longer period its effects were (positive and) significant in B.C., the prairies, and Quebec, but insignificant in Ontario and the Atlantic. Over the shorter period, the only change was that the effects in Ontario were also significant.

Introduction of 'time-squared" as a trend variable over the longer period did not change the effects of either economic indicator in any region and the time variable itself was significant everywhere except in the Atlantic. But including time over the shorter period had an effect different from when the time frame was shortened without it. With a time variable the shift changed the influence of inflation from positive to negative in Ontario (without time it hadn't changed) and yielded insignificant results in B.C. and the Atlantic (both had changed from positive to negative without a time variable). Only the changes in influence in the prairies and Quebec were unaffected by the addition of this time variable.

A new variable, the proportion of the male population that is between the ages of 15 and 29 - henceforth referred to as YM (young males) - was added to the model for homicide rates. The inclusion of this variable had a greater effect than did that of time. Over the years 1931-74 when YM was added, the effect of inflation changed from positive to negative in every region, the effect of unemployment changed from positive to negative in Ontario and the Atlantic, and from positive to insignificant in B.C., and changed the effect of increases in per capita income from insignificant to positive in Ontario and the Atlantic. The YM variable was significant everywhere except B.C.

Recall that, without the YM variable, the effects of inflation had been reversed from positive to negative in B.C., Quebec, and the Atlantic when the time period under analysis was shortened. Now, with the YM variable, and negative estimates of the effect of inflation in all regions over the longer period, restricting the time frame changes the inflation effect to positive in the prairies and Ontario. Recall also that the change in time span had changed the effects of unemployment from positive to negative in the prairies and Ontario, and from positive to insignificant in the Atlantic. But with the YM variable, the shortening of the time span altered the effect of unemployment on the homicide rate from positive to negative in the prairies and Quebec, and from negative to positive in the Atlantic. The rather critical influence that the addition of the YM variable has on the different effects of economic indicators through shortening the time span becomes harder to decipher when we see that the YM variable, although in the longer period significant everywhere but B.C., in the shorter period is significant **nowhere**.

FIGURE 6



4. First Admissions to Mental Hospitals

First admissions to mental hospitals decreased as the inflation rate rose over the period 1931-74 in all regions. In the post-war era, though, it increased with inflation in B.C. and the Atlantic and was no longer significantly related to inflation in the prairies.

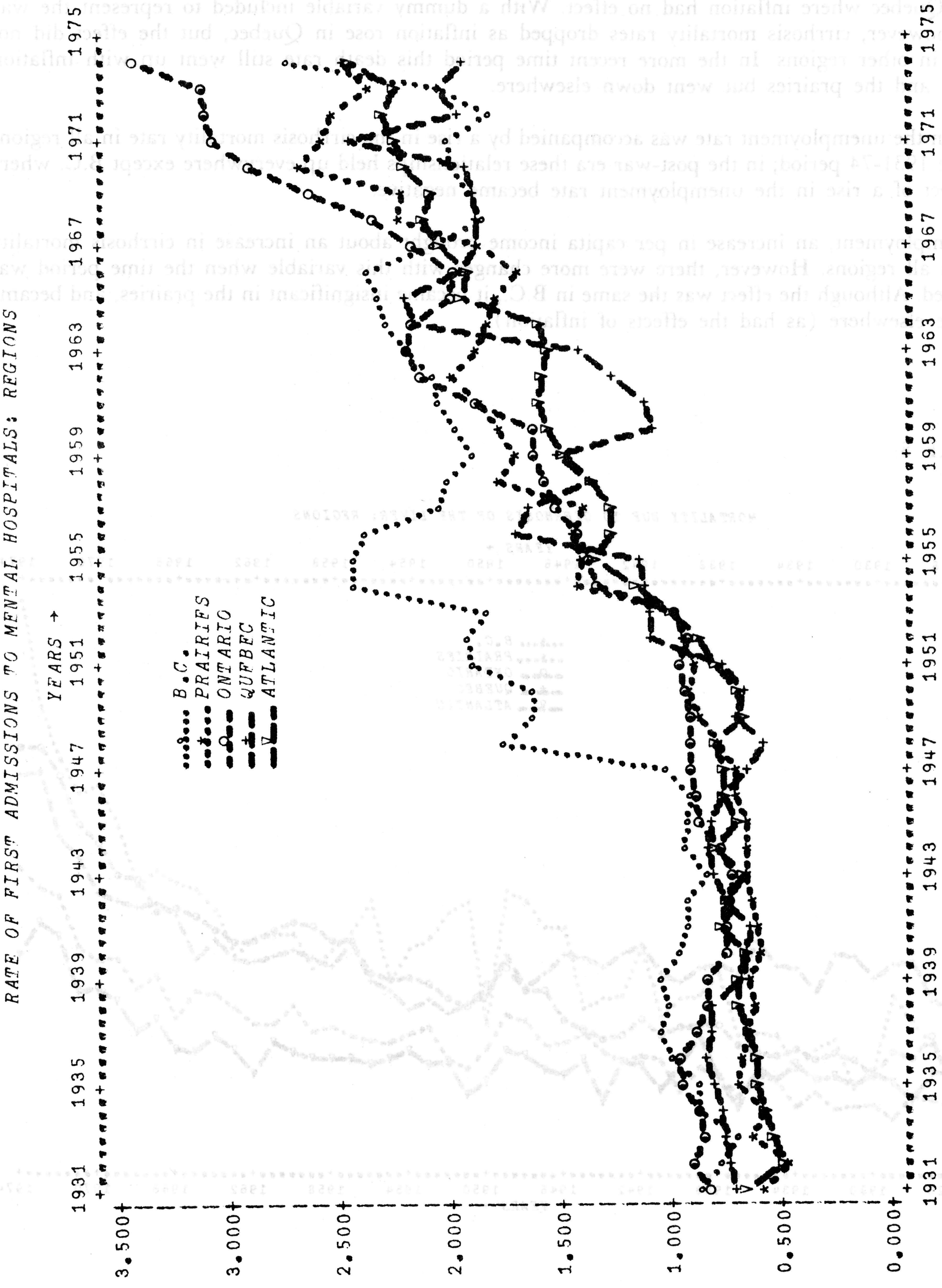
Increases in the unemployment rate brought about decreases in the rate of first admissions in B.C. and Quebec but increases in the prairies, Ontario, and the Atlantic. Excluding the earlier years from analysis, increases in unemployment still bring about decreases in Quebec, but now also decreases in the prairies and the Atlantic; the effect was to increase first admissions in B.C. and was insignificant in Ontario.

There was more stability in the relationship between first admissions and per capita income. A rise in per capita income brought about a rise in the rate of first admissions in all regions in the longer period and this relationship also held in the shorter period except in B.C. where it was reversed.

With per capita income removed and adding time variables, using "time-squared" rendered the effect of inflation in Ontario insignificant, while using "log-time" reduced its effect insignificant in B.C., the Atlantic, and again Ontario. The addition of "time-squared" changed the inflation effect from insignificant to negative in the prairies but from positive to insignificant in the Atlantic. But using "log-time" the effect in the prairies changed from insignificant to positive, from negative to positive in Ontario, and negative to insignificant in Quebec.

There was more sensitivity in the estimates of the unemployment effects. With "time-squared" (over the longer period) the influence of unemployment on first admissions changed from positive to negative in the prairies and the Atlantic. When "log-time" was used the unemployment effect went from negative to positive in B.C. and Quebec. In the post-war era the inclusion of "time-squared" made the unemployment effect in B.C. insignificant, changed it from negative to positive in the prairies, and insignificant to positive in Ontario. Using "log-time" had the same effect as "time-squared" except that the effect in B.C. remained significant.

FIGURE 7


RATE OF FIRST ADMISSIONS TO MENTAL HOSPITALS: REGIONS
YEARS →
1931
1935
1939
1943
1947
1951
1955
1959
1963
1967
1971
1975
3.500
3.000
2.500
2.000
1.500
1.000
0.500
0.000
B.C.
PRAIRIES
ONTARIO
QUEBEC
ATLANTIC

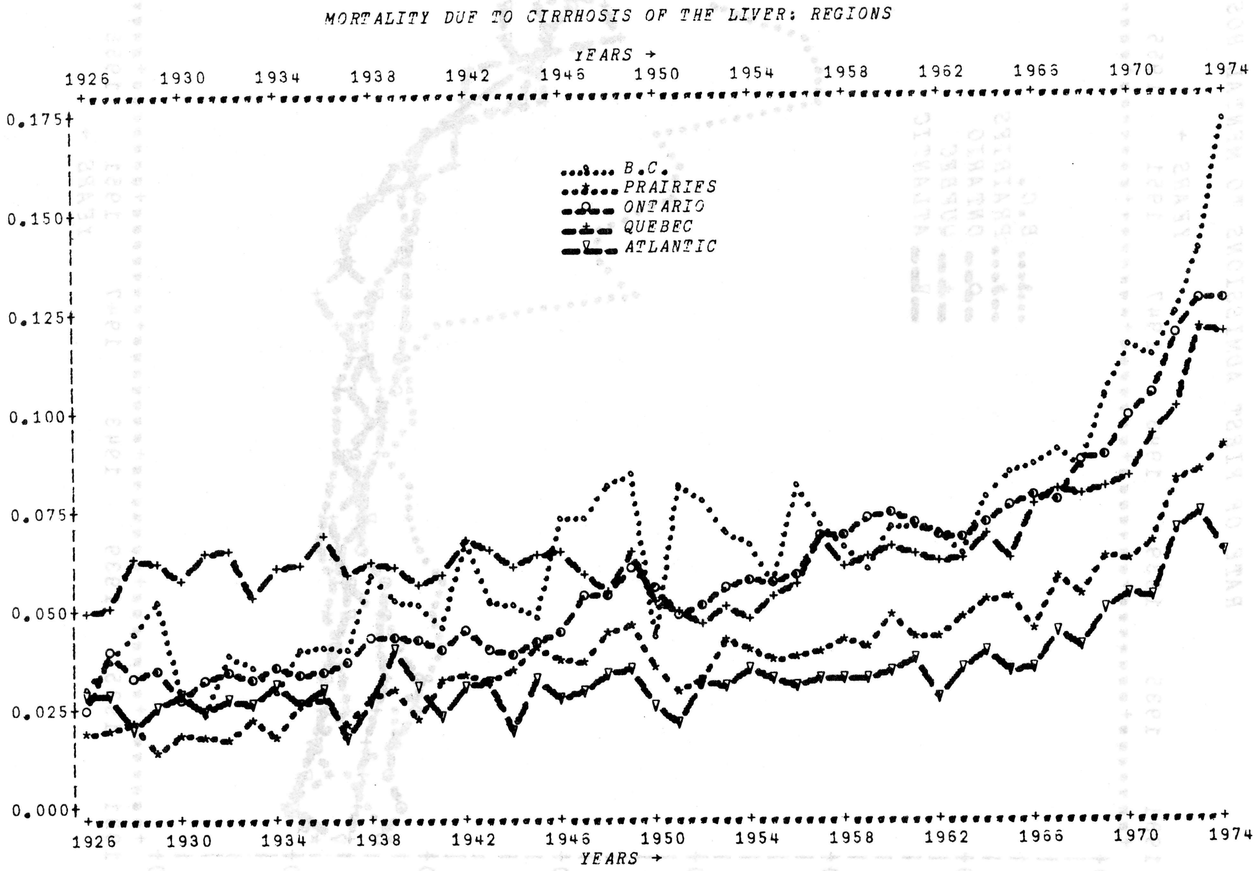
5. Deaths Due to Cirrhosis of the Liver

Deaths caused by cirrhosis of the liver increased with inflation over the longer period in all regions except Quebec where inflation had no effect. With a dummy variable included to represent the war years, however, cirrhosis mortality rates dropped as inflation rose in Quebec, but the effect did not change in other regions. In the more recent time period this death rate still went up with inflation in B.C. and the prairies but went down elsewhere.

A rise in the unemployment rate was accompanied by a rise in the cirrhosis mortality rate in all regions over the 1931-74 period; in the post-war era these relationships held up everywhere except B.C. where the effect of a rise in the unemployment rate became negative.

As unemployment, an increase in per capita income brought about an increase in cirrhosis mortality rates in all regions. However, there were more changes with this variable when the time period was shortened. Although the effect was the same in B.C., it became insignificant in the prairies, and became negative elsewhere (as had the effects of inflation).

FIGURE 8



Results by Age Group

1. Mortality

The mortality rate increased with inflation over the longer period. However, this relationship was maintained only in the 45-54 year group over the shorter period; with the youngest group it had no affect on mortality and with all other age groups mortality declined as inflation rose.

An increase in the unemployment rate brought about an increase in the mortality rate over 1931-74 for people in all age groups except those 55-64, for whom it was negative, and those over 75, for whom it had no effect. In the post-war period it still increased with unemployment for the two youngest age groups but decreased with unemployment for the next two (35-44, 45-54) and increased with unemployment for the three oldest. The change in sign from positive to negative for the middle age groups may reflect the reduced burden of unemployment in the past twenty years due both to unemployment insurance and the increase in the relative number of multi-earner families.

FIGURE 9 (a).

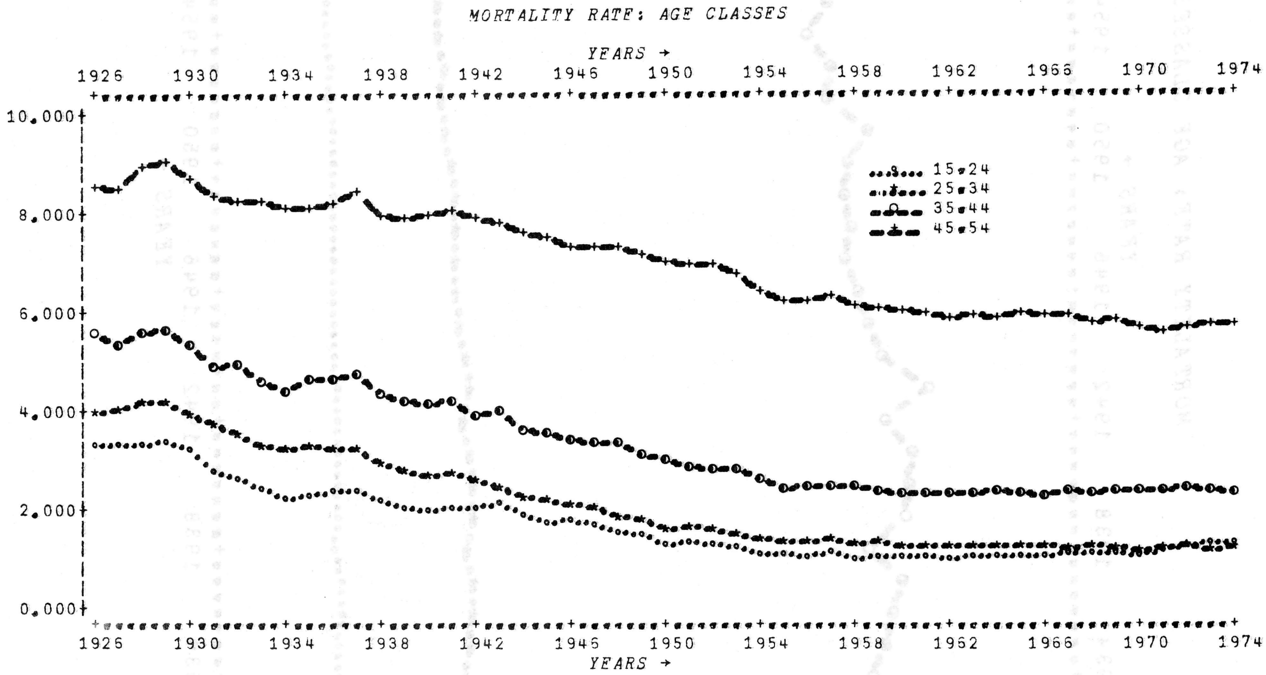
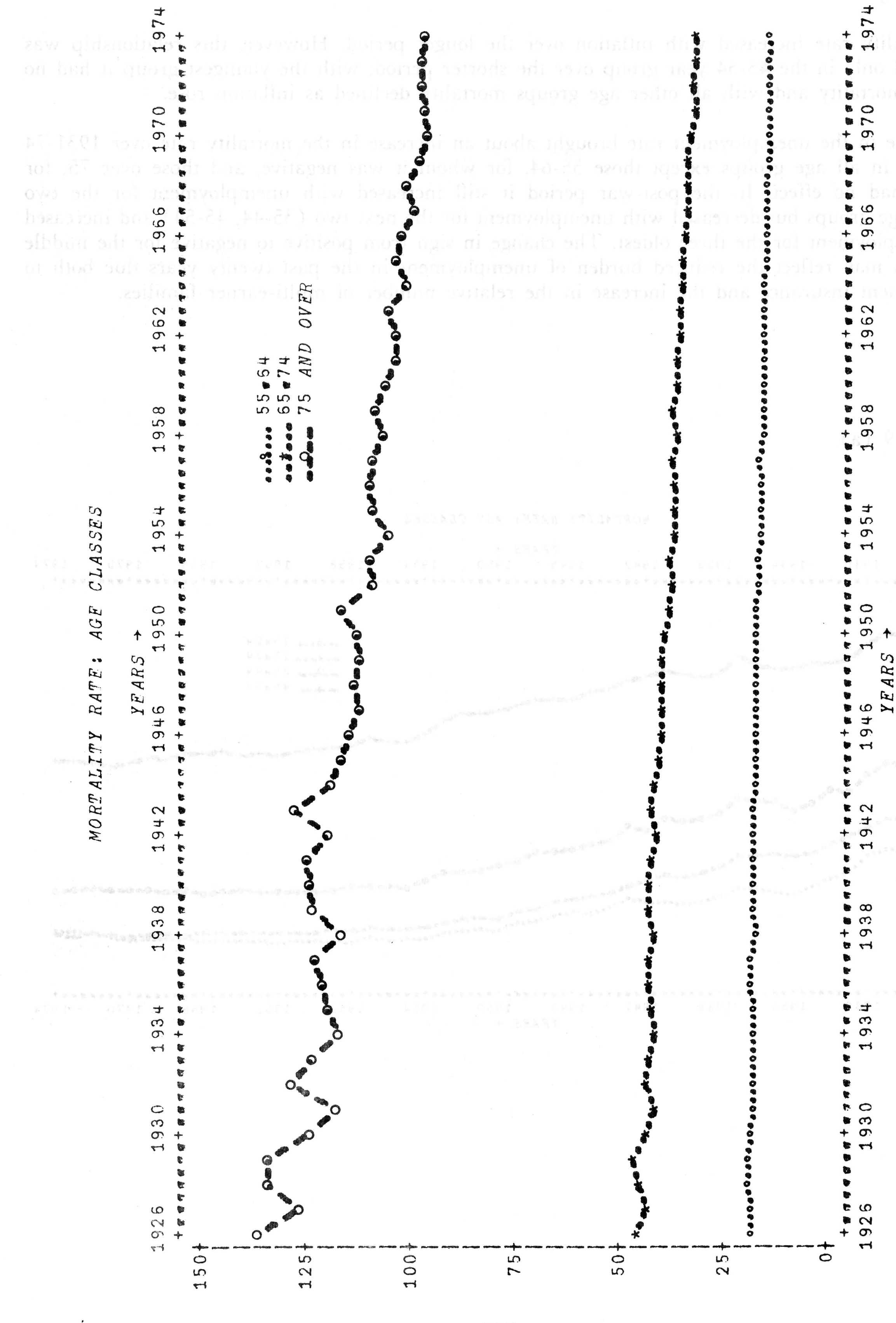


FIGURE 9 (b)



MORTALITY RATE: AGE CLASSES
YEARS →
1926
1930
1934
1938
1942
1946
1950
1954
1958
1962
1966
1970
1974
150
125
100
75
50
25
0
55-64
65-74
75 AND OVER

2. Suicide

Over the 1931-74 period inflation did not affect the suicide rate for the 25-34 or 55-64 age groups but it was associated with an increase in that rate for all other age groups. But in the post-war period all signs changed; now inflation has no effect on the youngest or second oldest age groups, but it brought about a decrease in the suicide rate for all other age groups.

The effects of unemployment on the suicide rate also varied across time periods. Over the longer period the rate went up with unemployment for all groups except the oldest for whom it declined; but in the shorter period it went down for all age groups.

FIGURE 10 (a)

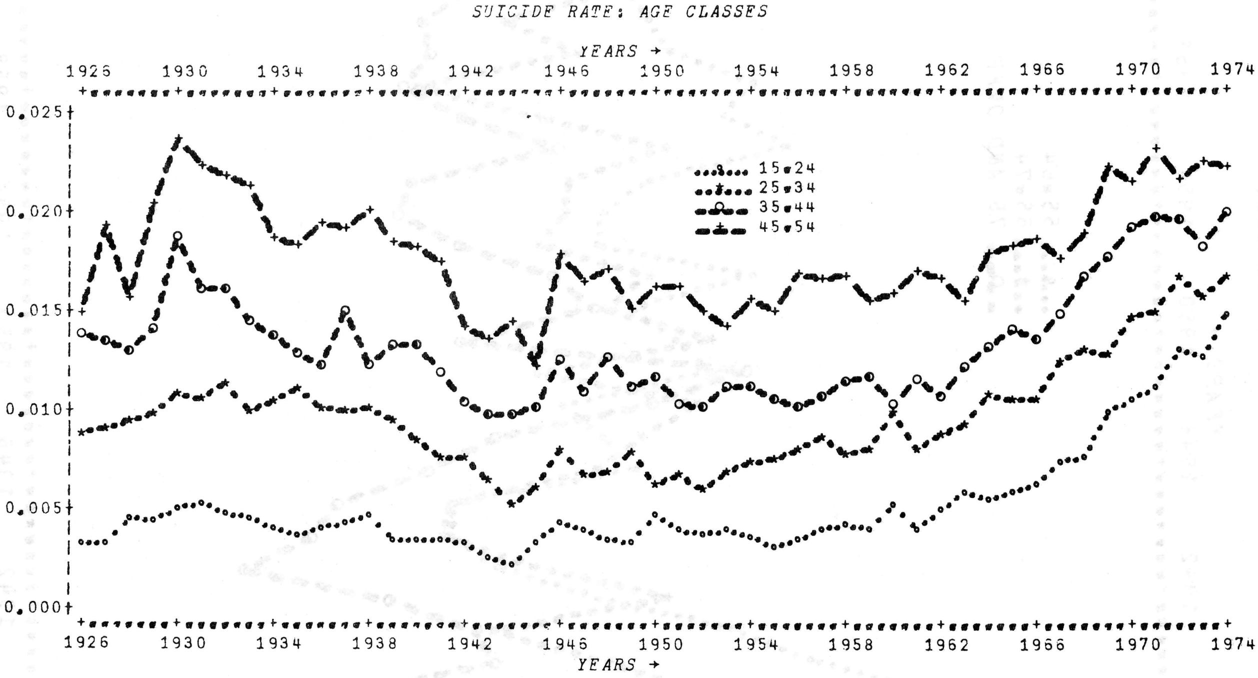
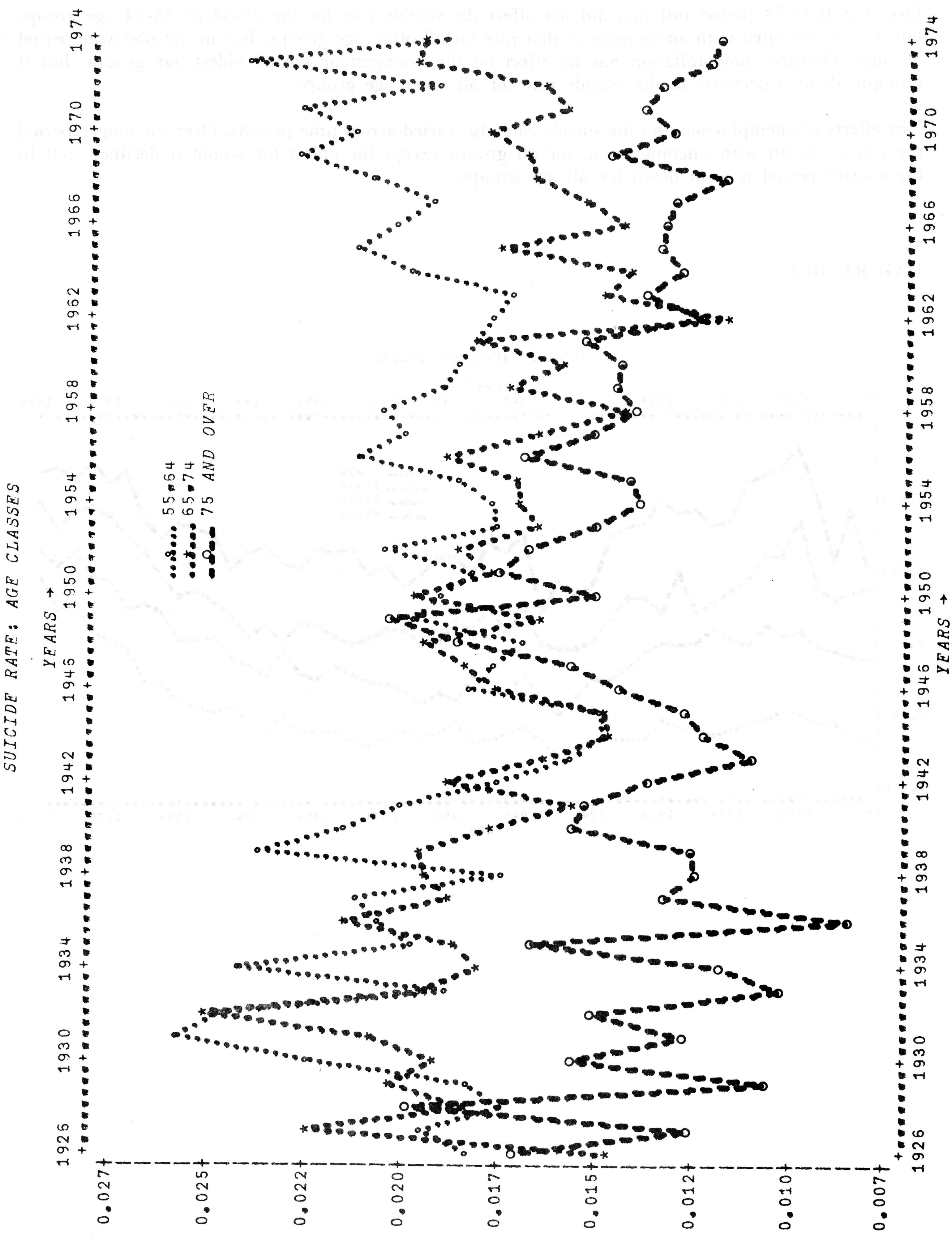


FIGURE 10 (b)


SUICIDE RATE: AGE CLASSES
YEARS →
1926
1930
1934
1938
1942
1945
1950
1954
1958
1962
1966
1970
1974
55-64
65-74
75 AND OVER
0.027
0.025
0.022
0.020
0.017
0.015
0.012
0.010
0.007
1926
1930
1934
1938
1942
1946
1950
1954
1958
1962
1966
1970
1974
YEARS →

3. First Admissions to Mental Hospitals

The rate of first admissions to mental hospitals decreased with inflation over the longer period for all groups except those in their 40's for whom inflation had no effect, and those in their teens, for whom inflation had a positive influence. Over the shorter period inflation had a positive effect on the rate of first admissions for the oldest group but a negative effect for all others.

From 1931-74 the rate declined with an increase in unemployment for those in their 30's, 40's and 50's, had no effect on the rate for those in their 60's but increased the rate for those in their 70's - and for those under 30. In the post-war period the rate of first admissions dropped with an increase in unemployment for all age groups except the oldest for whom it still increased.

FIGURE 11 (a)

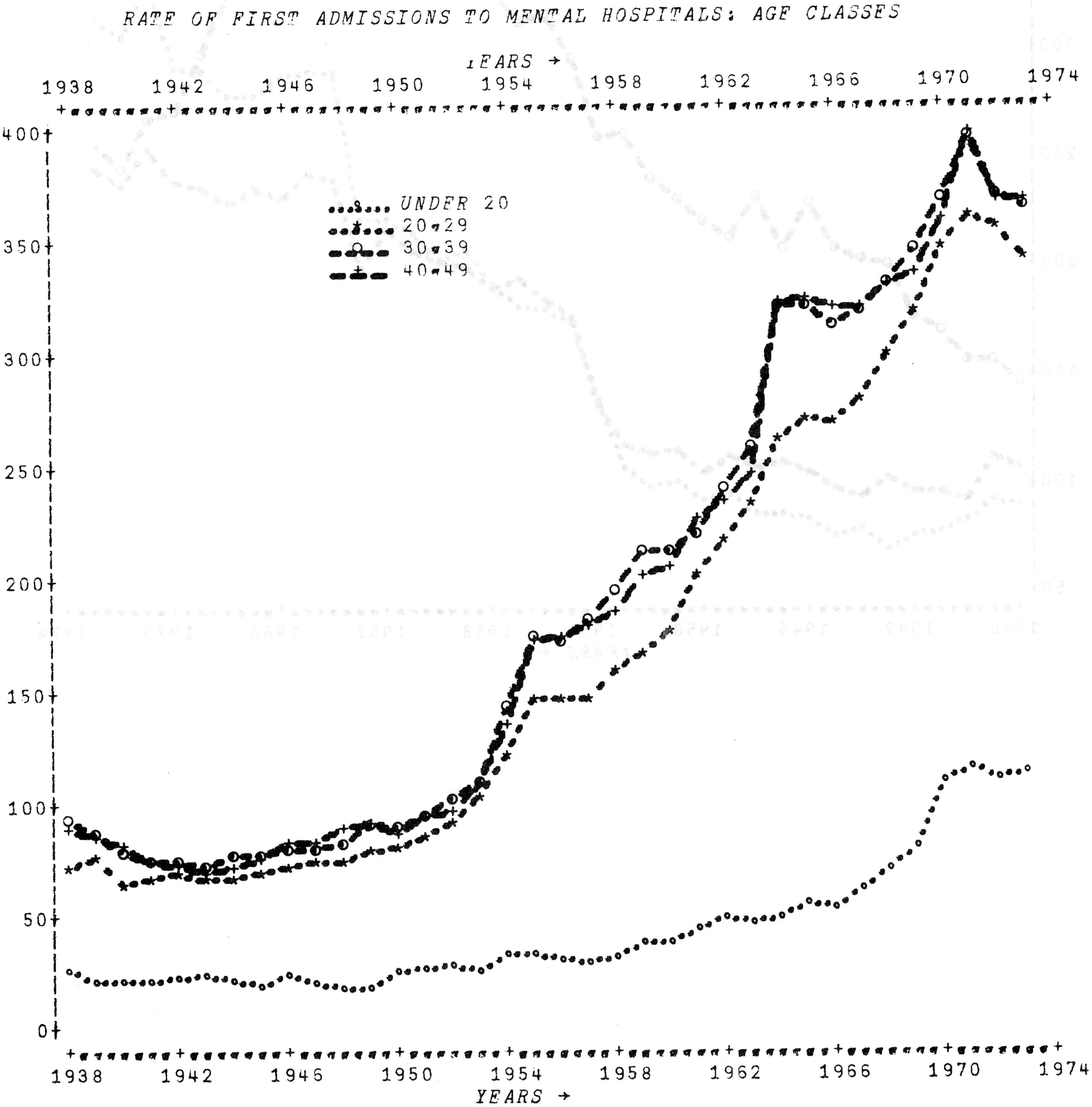
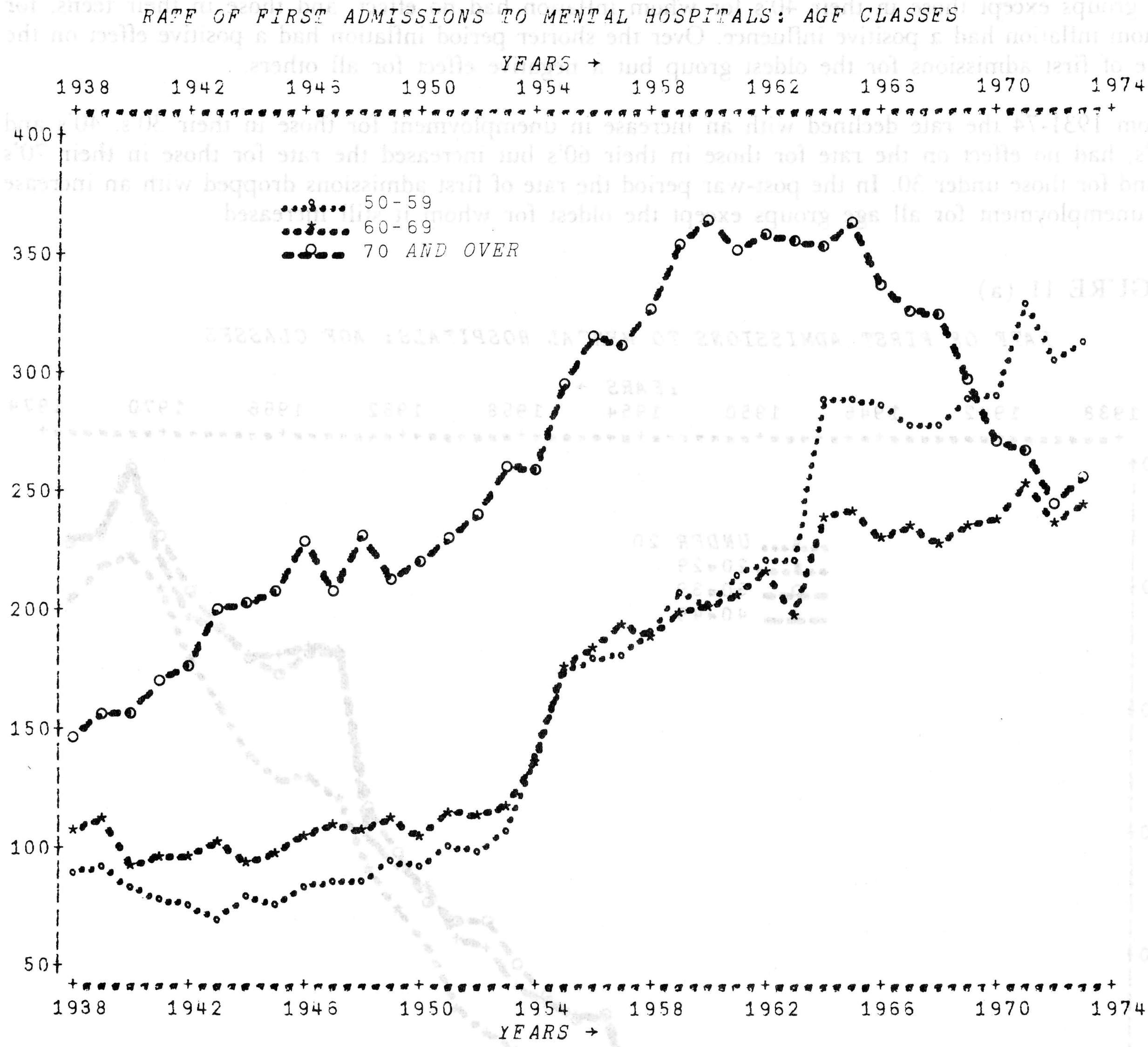


FIGURE 11 (b)


RATE OF FIRST ADMISSIONS TO MENTAL HOSPITALS: AGE CLASSES
YEARS →
1938
1942
1946
1950
1954
1958
1962
1966
1970
1974
400
350
300
250
200
150
100
50
50-59
60-69
70 AND OVER
YEARS →

4. Rate of Convictions for Indictable Offences

The rate of convictions increased with inflation for all age groups over the longer period; the only change over the shorter period was that it decreased for the youngest group.

Again, possibly reflecting the declining severity of unemployment, convictions increased with this rate over the longer period for all age groups and, just as uniformly, declined in the post-war period.

FIGURE 12 (a)

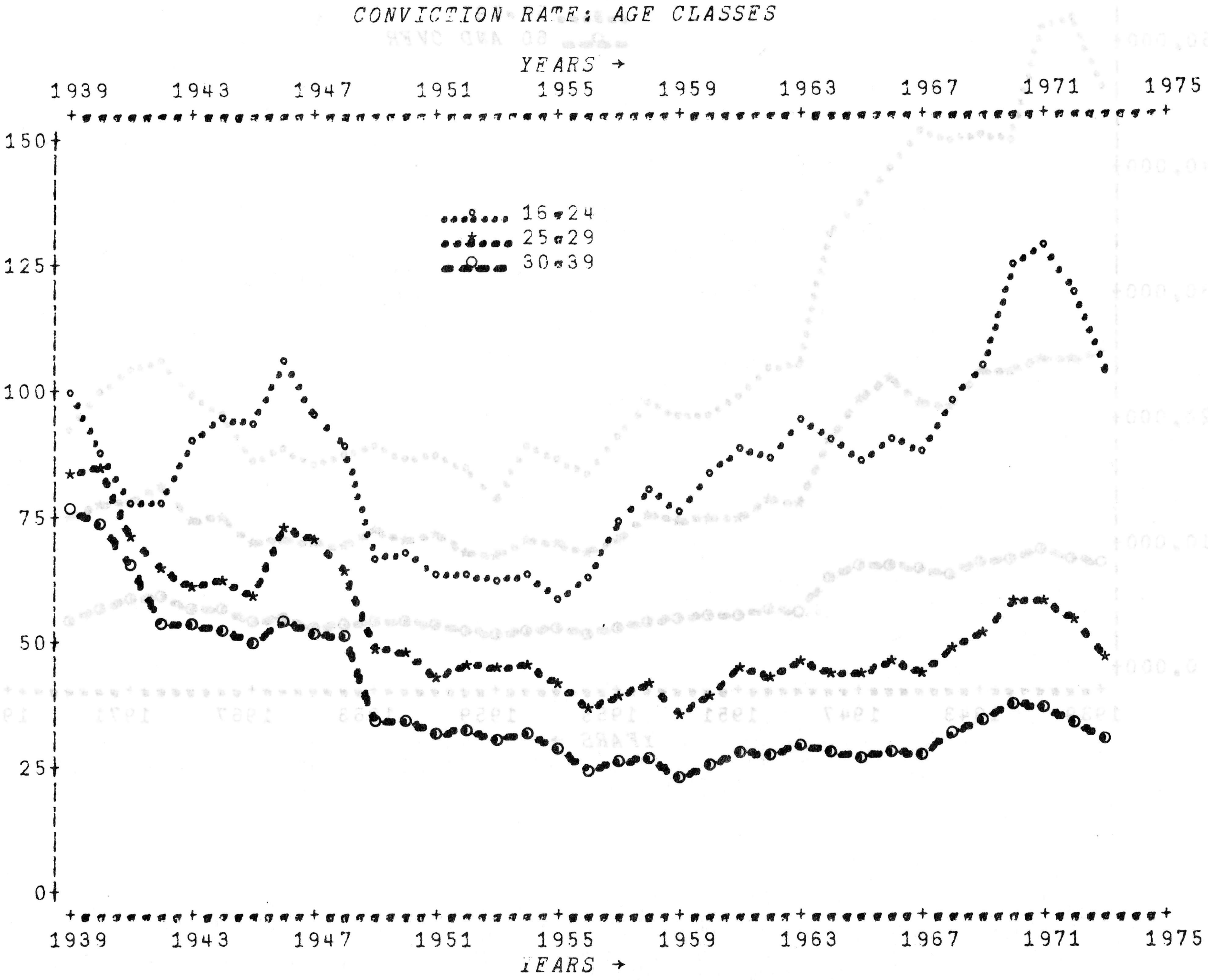
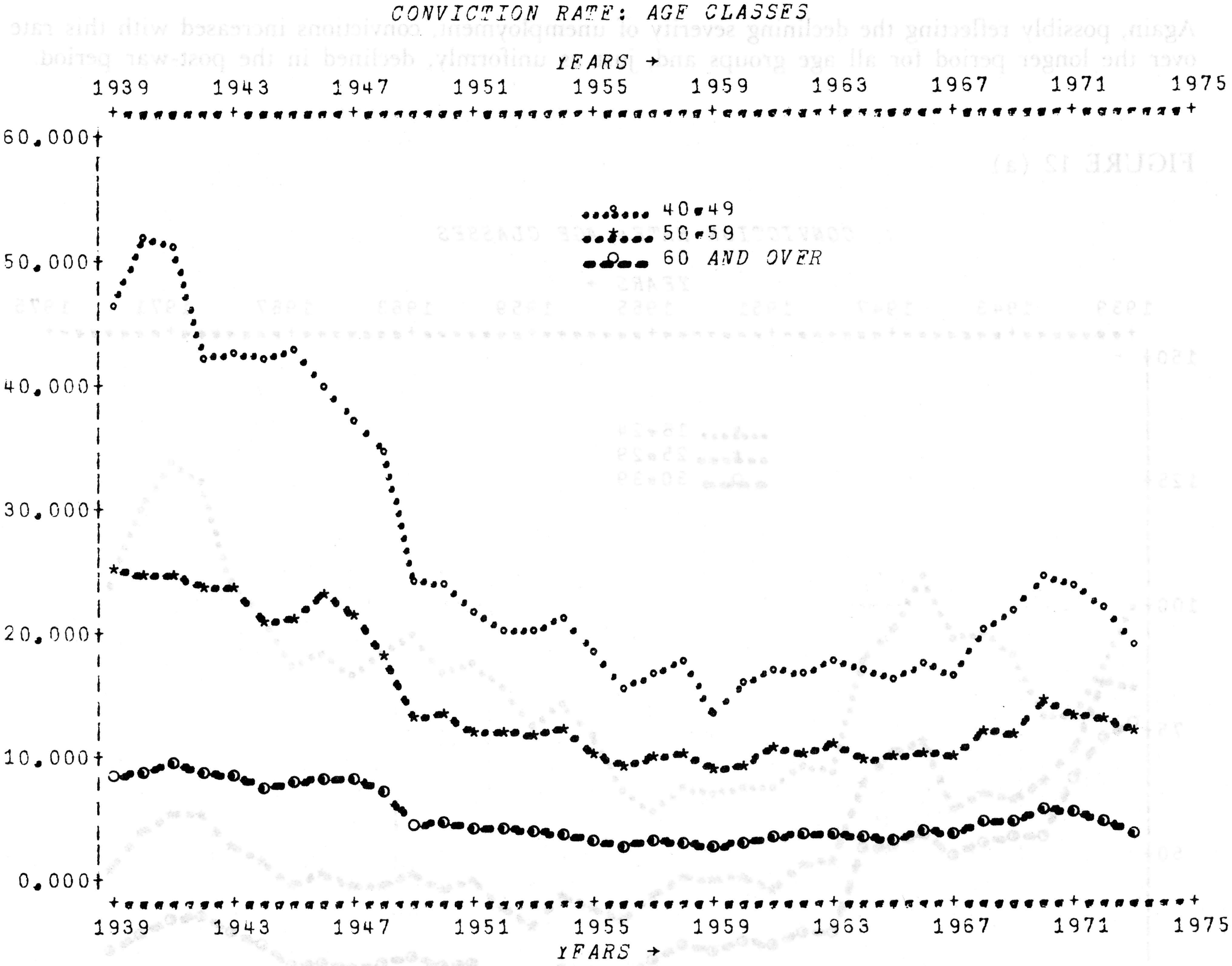


FIGURE 12 (b)


CONVICTION RATE: AGE CLASSES
YEARS →
1939
1943
1947
1951
1955
1959
1963
1967
1971
1975
60,000
50,000
40,000
30,000
20,000
10,000
0,000
40-49
50-59
60 AND OVER
YEARS →

Direction of Effects of Independent Variables (By Region)

a) Mortality Rate

	Inflation					Unemployment				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
L	+	+	+	+	+	–	–	–	+	+
LD	+	+	+	+	+	–	–	–	+	+
S	–	0	0	0	–	+	+	+	+	+

b) Suicide Rate

	Inflation					Unemployment				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
L	+	+	+	+	+	+	+	+	+	+
LD	+	+	+	+	+	+	+	+	+	+
LT_1	+	+	+	+	+	+	+	+	+	+
LT_2	+	+	+	+	+	+	+	+	+	+
LT_1D	+	+	+	+	+	+	+	+	+	+
LT_2D	+	+	+	+	+	+	+	+	+	+
S	–	+	0	–	–	–	–	–	–	–
ST_1	+	+	+	+	+	–	–	–	–	–
ST_2	–	+	–	–	–	–	–	–	–	–

c) Homicide Rate

	Inflation					Unemployment				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
L	+	+	+	+	+	+	+	+	+	+
LD	0	+	+	+	0	+	+	+	+	+
LP	−	−	−	−	−	+	+	−	+	−
LT_2	+	+	+	+	+	+	+	+	+	+
LPD	−	−	−	−	−	+	+	−	+	−
LT_2D	0	+	+	+	+	+	+	+	+	+
S	−	+	+	−	−	+	−	−	+	0
SP	−	+	+	−	−	+	−	−	0	+
ST_2	0	+	−	−	0	+	0	−	+	−

d) First Admission to Mental Hospitals

	Inflation					Unemployment				
	(1)	(2)	(3)	(4)	(5)	(1)	(2)	(3)	(4)	(5)
L	−	−	−	−	−	−	+	+	−	+
LD	0	−	−	−	−	−	+	+	−	+
LT_1	0	−	0	−	0	+	+	+	+	+
LT_2	−	−	0	−	−	−	−	+	−	−
LT_1D	0	−	0	−	0	0	+	+	+	+
LT_2D	0	−	−	−	−	−	−	+	−	−
S	+	0	−	−	+	+	−	0	−	−
ST_1	+	+	+	0	+	+	+	+	−	−
ST_2	+	−	−	−	0	0	+	+	−	−

e) Deaths Due to Cirrhosis of the Liver

Inflation

	(1)	(2)	(3)	(4)	(5)
L	+	+	+	0	+
LD	+	+	+	–	+
S	+	+	–	–	–

Unemployment

(1)	(2)	(3)	(4)	(5)
+	+	+	+	+
+	+	+	+	+
–	+	+	+	+

Key: (1) = B.C.
(2) = Prairies
(3) = Ontario
(4) = Quebec
(5) = Atlantic

L = long period (1931-74)
S = short period (1955-74)
T_1 = log-time as trend
T_2 = time squared as trend
D = dummy variable for war years (1939-45)

Direction of Effects of Independent Variables (By Age)

a) Convictions Rate

Inflation

	16-24	25-29	30-39	40-49	50-59	60+
1939-73	+	+	+	+	+	+
1955-73	-	+	+	+	+	+

Unemployment

16-24	25-29	30-39	40-49	50-59	60+
+	+	+	+	+	+
-	-	-	-	-	-

b) Mortality Rate

Inflation

	15-24	25-34	35-44	45-54	55-64	65-74	75+
1931-74	+	+	+	+	+	+	+
1955-74	0	-	-	+	-	-	-

Unemployment

	15-24	25-34	35-44	45-54	55-64	65-74	75+
1931-74	+	+	+	+	-	+	0
1955-74	+	+	-	-	+	+	+

c) First Admissions to Mental Hospitals

Inflation

	0-19	20-29	30-39	40-49	50-59	60-69	70+
1943-73	+	-	-	0	-	-	-
1955-73	-	-	-	-	-	-	+

Unemployment

	0-19	20-29	30-39	40-49	50-59	60-69	70+
1943-73	+	+	-	-	-	0	+
1955-73	-	-	-	-	-	-	+

d) Suicide Rate

Inflation

	15-24	25-34	35-44	45-54	55-64	65-74	75+
1931-74	+	0	+	+	0	+	+
1955-74	0	-	-	-	-	0	-

Unemployment

	15-24	25-34	35-44	45-54	55-64	65-74	75+
1931-74	+	+	+	+	+	+	-
1955-74	-	-	-	-	-	-	-

Commentary

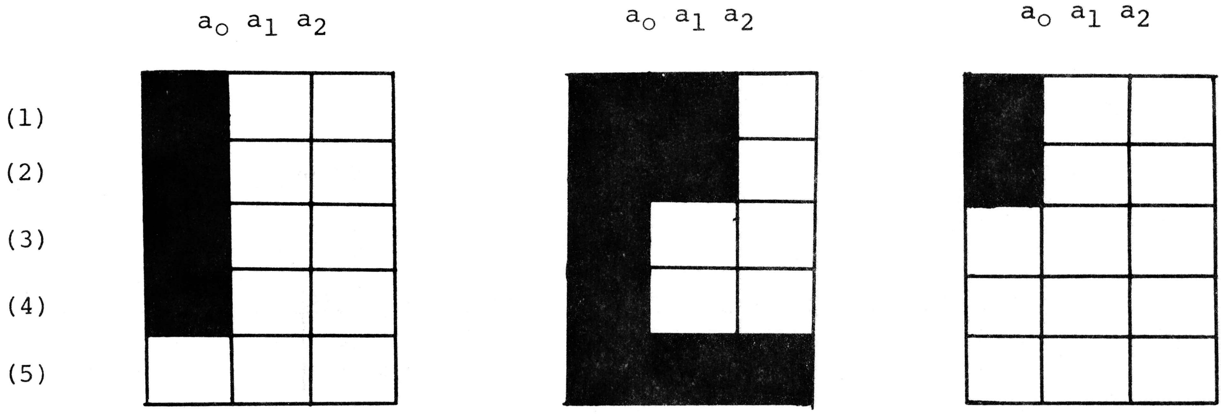
The polynomial distributed lag method has been applied to several data sets representing variables in the social sciences. Judging from the results, the method must be considered at best highly sensitive to points which we would hope would have to been minor, and at worst, it must be considered meaningless. (This, of course, is not a criticism of the APL package which allows the method to be used.) The PDL method has its origins in investment models where the length of time (say the number of quarters) that one factor influences another is more or less certain. In the social sciences such certainty seldom, if ever, exists.

This skepticism of the applicability of the PDL method to subjects as the social sciences arises not only from the oftentimes unexpected change in the direction of effects of the independent variables but also from two further empirical exercises using the same data which are discussed below.

One of the ways of assessing the appropriateness of the lag structure is by testing the significance of the coefficients of the polynomial. In the charts below the significance of the coefficents from the regional analysis are shown by a shaded area; an unshaded area indicates the coefficient was insignificant.

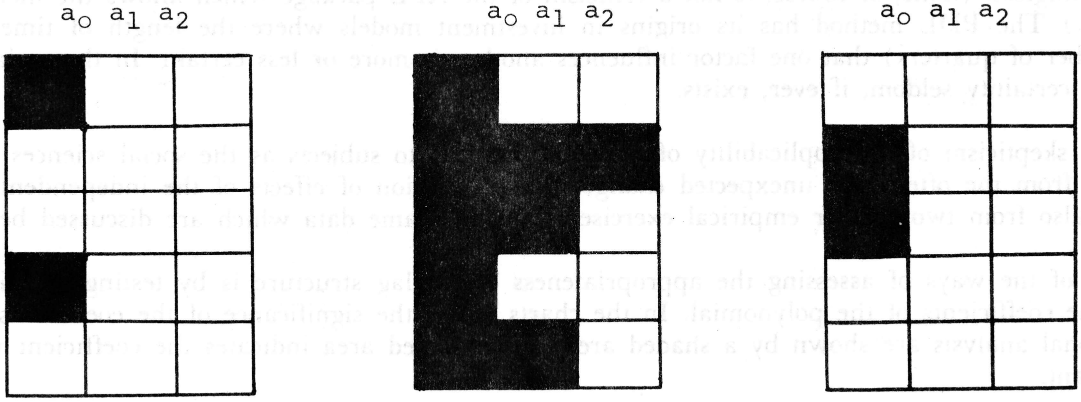
Significance of Polynomial Coefficients

1. Mortality 1931-74



Region	Inflation a_o	Inflation a_1	Inflation a_2	Unemployment a_o	Unemployment a_1	Unemployment a_2	Per Capita Income a_o	Per Capita Income a_1	Per Capita Income a_2
(1)	▨			▨	▨		▨		
(2)	▨			▨	▨		▨		
(3)	▨			▨					
(4)	▨			▨					
(5)				▨	▨	▨			

2. Suicide 1931-74



Region	Inflation a_0	Inflation a_1	Inflation a_2	Unemployment a_0	Unemployment a_1	Unemployment a_2	Per Capita Income a_0	Per Capita Income a_1	Per Capita Income a_2
(1)	■			■					
(2)				■	■	■	■		
(3)				■	■		■		
(4)	■			■					
(5)				■	■				

Key:
(1) = Atlantic
(2) = Quebec
(3) = Ontario
(4) = Prairies
(5) = British Columbia

3. Homicide 1931-74

Region	Inflation			Unemployment			Per Capita Income		
	a_o	a_1	a_2	a_o	a_1	a_2	a_o	a_1	a_2
(1)									
(2)				■					
(3)				■					
(4)				■					
(5)				■					

4. First Admissions to Mental Hospitals 1931-74

Region	Inflation			Unemployment			Per Capita Income		
	a_o	a_1	a_2	a_o	a_1	a_2	a_o	a_1	a_2
(1)	■								
(2)								■	■
(3)	■							■	■
(4)	■	■							
(5)									

A discussion of the above in light of what was reported in Section 3 is illuminating. In that section, mortality was found to have increased significantly with inflation in all regions. Yet as the chart shows, the coefficients of the linear and quadratic term were never significant and in British Columbia not even the constant (in the polynomial) was significant. A few more of the coefficients corresponding to the unemployment variable were significant but virtually none of the per capita income coefficients were, even though the β's derived from them were in all regions.

Similar results were found with suicide as the dependent variable. With homicide only the constant term was significant in assessing the effects of unemployment, but not even that in the Atlantic region. Moreover, **no** coefficients for inflation or per capita income were significant anywhere, yet from the earlier results the latter was significant everywhere but Ontario and the Atlantic, and inflation was significant everywhere. As for first admissions Section 3 showed the unemployment rate to have a significant influence in all regions, yet again, none of the polynomial coefficients were significant in any region.

There is additional uncertainty as to how to interpret the results when one examines the values of the coefficients of a lagged variable in relation to the sum of those coefficents; in other words, how does the "total" effect of an independent variable compare with its year-by-year effects? And the key issue is how we interpret the significance of the total effect versus the significance of the yearly effects. The interpretation of the results becomes problematic when, say with a five year lag, only one or two of the yearly coefficients are significant and under the hypothesized five year lag the total effect is significant. Intuitively one would want the fact that most of the yearly coefficients are insignificant to dominate the significance decision of the total effect. Indeed, in more than one instance in the study, the total effect was significant while **none** of the yearly coefficients were.

This latter occurrence could only be justified with an explanation that, while an independent variable has little influence on a yearly basis, over a five year period the effects of that variable accumulate and actually do in total exert a significant influence. More likely, however, is that the lag specification is incorrect. Although no hard and fast criteria can be applied as to the suitability of the lag structure, it is worthwhile in closing, to offer some suggested criteria for evaluating the choice of the structure.

Suggested Criteria for Evaluating Lag Structure

1) If the leading coefficient of the polynomial is insignificant reduce its degree and re-estimate the model. Continue this procedure until the leading coefficient is significant. If even a linear polynomial proves unnecessary then no lag is needed. Although it is theoretically possible that while, say, the leading coefficient in a cubic is not significant, that of a quartic would have been; in none of our work has the coefficient of a term in a polynomial been insignificant while that of the next highest was significant. Thus, it is probably unlikely that this would occur.

2) Having settled on the degree of the polynomial, the length of the lag should be based on the significance of the annual β coefficients. If the "least recent" coefficient is insignificant reduce the lag length and re-estimate the model. Continue this procedure until the "least recent" β coefficient is significant. A similar procedure can be applied to the "most recent" β because the optimal lag length may not include a value of the independent variables for the year concurrent with the value of the dependent variable. A problem arises when the most and least recent β's are significant but one or more of the intermediate β's are not. There are obviously two choices: leave the intermediate value in with the justification that the effect of the variable subsided but then became strong again (or changed from positive to negative or vice-versa) and the total effect should include this transitional period; or delete the value for that year, which would involve some re-working of the estimating equations.

These two criteria are obviously not the only criteria to be used; they should be considered as criteria additional to the other statistics (R^2, F, Durbin-Watson, etc.) used in evaluating ordinary least-squares estimation procedures.

Conclusion

The polynomial distributed lag method should be applied with caution in estimating relationships among variables from social science disciplines. The method was originally developed to deal with relationships in investment models wherein there is much more certainty as to the suitability of the lag structure. The first choice should be no lag structure, the coefficients should be estimated directly. When degrees of freedom are a problem a lag structure can be imposed but careful attention must be paid to the significance of the polynomial coefficients and the β coefficients before the results can be interpreted in a meaningful way. The results also appear to be very sensitive to the time period over which the variables are observed.

Acknowledgements

This paper, although written by Neil MacLeod, represents the joint effort of both he and Normand Briere, also of the Long Range Planning Directorate of National Health and Welfare.

References

1. Foot, D. and A. North, **The Use and Misuse of Econometrics (with Reference to Sharp APL)**, Institute for Policy Analysis, University of Toronto, 1977.

2. Johnston, J., **Econometric Methods**, McGraw-Hill, New York, 1972.

3. Statistics Canada, "Vital Statistics", Catalogue No. 84-201, 84-206, Ottawa, 1925-75.

Printed in
Canada